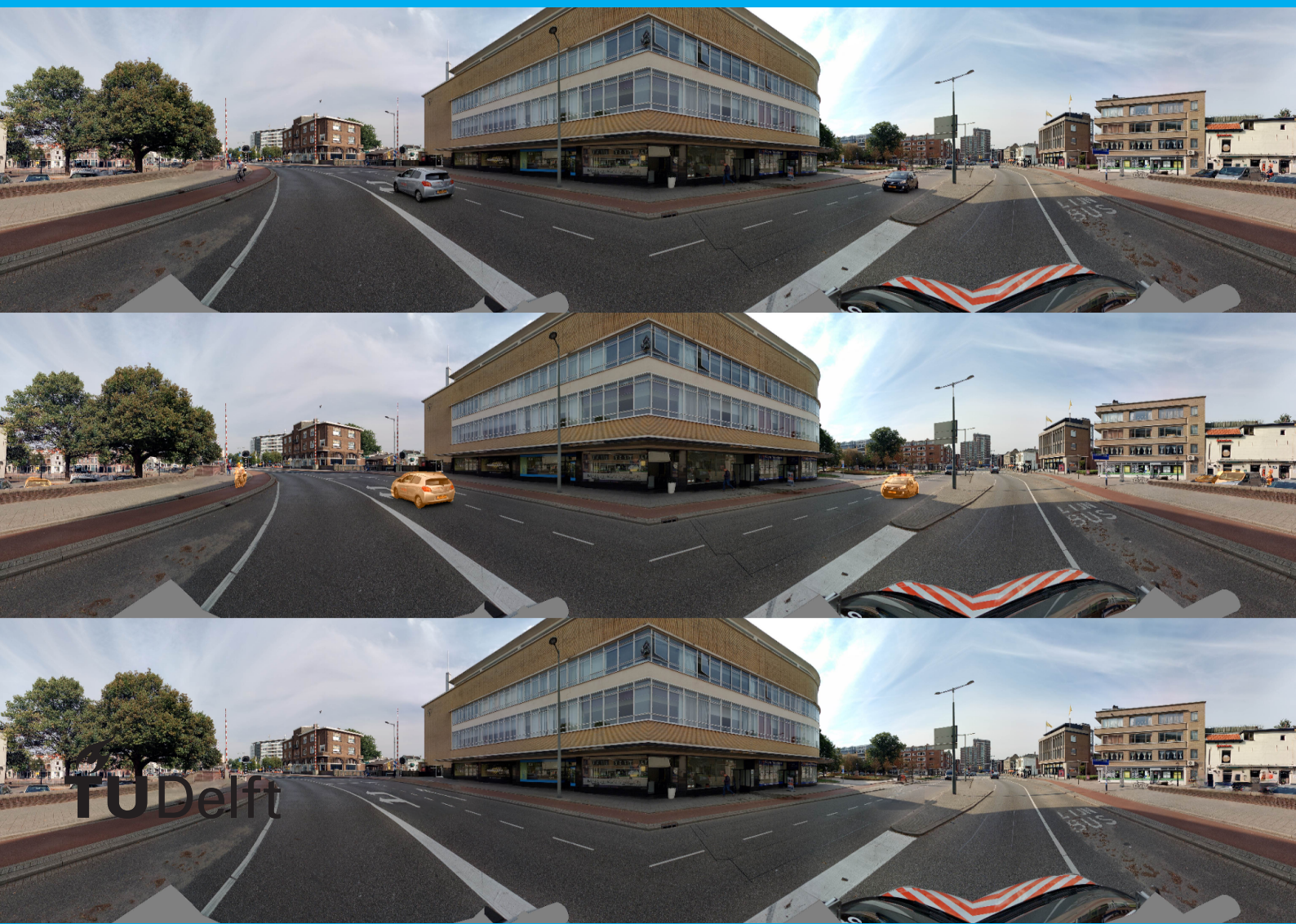
Moving object detection and image inpainting in street-view imagery

M. C. Uittenbogaard

Thesis report for the MSc Programme in Mechanical Engineering at TU Delft
Performed at CycloMedia, Zaltbommel



TU Delft

Moving object detection and image inpainting in street-view imagery

by

M. C. Uittenbogaard

to obtain the degree of Master of Science
at the Delft University of Technology,
to be defended publicly on Friday September 28, 2018 at 12:45

Student number:	4314298	
Project duration:	December 1, 2017 – September 1, 2018	
Thesis committee:	prof. dr. Dariu Gavrila	TU Delft, supervisor
	dr. Julian Kooij	TU Delft
	dr. Roderik Lindenbergh	TU Delft

An electronic version of this thesis is available at `http://repository.tudelft.nl/`.

TU Delft

Preface

Before you lies the thesis 'Moving object detection and image inpainting in street-view imagery', which consists of a research into the fields of image inpainting and moving object detection and the creation of a fully automated pipeline which combines those two fields to remove moving objects from street-view imagery. This thesis is written to graduate from the master of Mechanical Engineering at the Delft University of Technology (TU Delft), with the track Vehicle Engineering. The research and development that lead to this thesis was done from December 2017 to September 2018.

This thesis was done externally at CycloMedia, a supplier of street-view imagery. The head office and Research and Development department of CycloMedia is located in Zaltbommel, which is also the location at which this thesis was written. It was a tough task to measure up to all the requirements as this research is very broad, consisting of two separate assignments. However, the final outcome does satisfy the requirements. A working pipeline is delivered which is able to find and remove moving objects from street-view imagery without requiring any human intervention.

Reaching this final result would not have been possible without the help of all my supervisors. I would like to thank my daily supervisors at CycloMedia, J. Vijverberg and C. Sebastian, for their day-to-day guidance and support. I would also like to thank my supervisor from the TU Delft, D. Gavrila, for supporting me and guiding me through the process of graduating from the TU Delft.

Finally I would like to thank all my other colleagues during my time at CycloMedia, mainly the Research and Development team 'Data Analytics'. They were always available to give suggestions, discuss my ideas and help me with my problems regarding the coding and the CycloMedia data and network structure. Next to all their help, they also made my time at CycloMedia a pleasant experience.

Note that, as many results are shown as a color overlay, it is recommended to read this thesis in color. I hope you enjoy your reading.

M. C. Uittenbogaard
Delft, September 2018

Contents

1 Introduction **1**
- 1.1 Context . . . 1
- 1.2 Research questions . . . 3
- 1.3 Contributions . . . 4
- 1.4 Outline . . . 4

2 Literature Study **5**
- 2.1 Introduction to Neural Networks . . . 5
- 2.2 Moving object detection . . . 9
  - 2.2.1 Image segmentation . . . 10
  - 2.2.2 Utilization of CycloMedia LIDAR data . . . 12
- 2.3 Image inpainting . . . 13
  - 2.3.1 Exemplar-based inpainting . . . 14
  - 2.3.2 Multiple view inpainting . . . 15
  - 2.3.3 Generative Adversarial Networks . . . 16
  - 2.3.4 Comparison . . . 17

3 Methodology **19**
- 3.1 Moving object detection . . . 19
  - 3.1.1 Image segmentation . . . 19
  - 3.1.2 Re-projection . . . 23
  - 3.1.3 Compare re-projections with the image of interest . . . 28
  - 3.1.4 Combine segmentation and moving object detection results . . . 28
- 3.2 Image inpainting . . . 31
  - 3.2.1 Segregate objects . . . 31
  - 3.2.2 Extracting background information . . . 31
  - 3.2.3 Generative Adversarial Network . . . 31

4 Experiments **35**
- 4.1 Moving object detection . . . 35
  - 4.1.1 Image segmentation . . . 35
  - 4.1.2 Image re-projection . . . 39
  - 4.1.3 Comparing different views . . . 41
  - 4.1.4 Combining of segmentation with pixel-level moving object detection . . . 46
- 4.2 Image inpainting . . . 47
  - 4.2.1 Creating the input of the inpainting network . . . 47
  - 4.2.2 The image inpainting network . . . 50

5 Discussion **55**

6 Conclusion **57**

Bibliography **59**

Appendices **63**

A Survey questions **63**
- A.1 Correct tiles - Tiles in which a moving object was removed . . . 63
- A.2 Incorrect tiles - Tiles in which no moving object was removed . . . 68

B Various pipeline outputs **71**

C Sequences of Cycloramas **75**

1 Introduction

The detection of moving objects is an important task with many applications. One notable application is the field of self-driving vehicles. For the safety of drivers and other (vulnerable) road users, the detection of other road users is of high importance. Many Advanced Driver-Assistance Systems (ADAS), for instance, Adaptive Cruise Control (ACC) and Emergency Brake Assist (EBA), rely on the perception of other road users. Good detection of other road users is an important step towards the goal of improving traffic safety. Furthermore, removing these objects from the data is desirable in some cases. This could be, for instance, of interest for the creation of high-detail roadmaps. For suppliers of high-detail street-view imagery it could, for instance, be used as a privacy measure, as it could replace blurring faces and license plates. It is a difficult challenge, because after removing these objects from image data a hole is created in the image. This hole should be filled with the actual content of the scene behind the moving object, to keep the data truthful. Besides the requirement of truthful image data, the images with removed moving objects should also look natural without many artifacts.

Those two challenging fields of research, moving object detection and image inpainting, are combined in this thesis, performed at CycloMedia. This company, based in Zaltbommel, supplies high-detail street-view imagery combined with LIDAR point clouds and highly accurate positioning. The goal of this thesis is to create a fully automated pipeline which removes moving objects from image data and to make an inpainting of the hole which occurs when removing data from images. To do this, LIDAR point clouds and image data at a 5-meter interval are available. In this introduction, some information about the company at which this thesis is performed and the available data is given first. Secondly, the research questions of this thesis are stated. After that, the contributions of this thesis to the state-of-the-art in moving object detection and image inpainting are presented. This section ends with a short explanation of the outline of this thesis.

1.1. Context

CycloMedia, based in Zaltbommel, is a supplier of street-view and aerial imagery and LIDAR point clouds. In this thesis, the street-view data is the main focus. This data is recorded yearly and with high accuracy, which makes it interesting for customers to perform tasks like virtual inspection, inventory and infrastructural planning, without visiting the location physically. The recorded data is sold in the form of an online viewer, StreetSmart, in which the imagery and point clouds can be found. In this viewer, many useful functions are available, for instance measuring distances or surfaces on the maps or the image data. Customers are for instance municipalities, utilities, and insurance companies. CycloMedia provides full coverage in the Netherlands and partly covers other countries like France, Germany, the United States and Scandinavian countries. For this thesis I was a part of the Research Development department, which is located in the home office in Zaltbommel.

This image and LIDAR data is recorded with the Digital Cyclorama Recorder (DCR) system. This system, which is mounted on top of the recording cars (figure 1.1c), is visible in figure 1.1a. It consists of five cameras with their focal points on one line, parallel to the driving direction. While driving, a parallax-free 360°

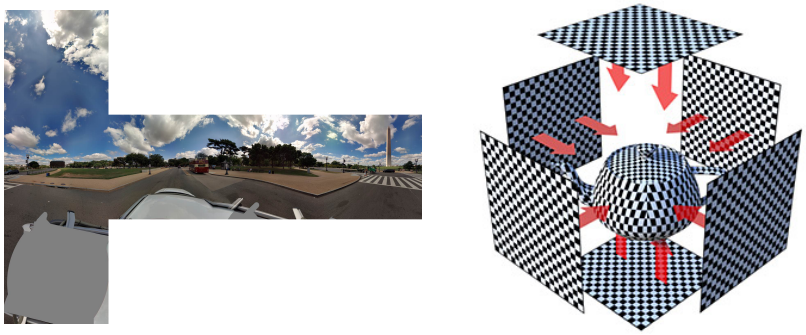
panoramic image (Cyclorama) can be taken by triggering the five cameras with a time interval, making sure the focal point of the images is at exactly the same spot. This method of creating parallax-free images is patented by CycloMedia [1]. These images, with a resolution of 100 megapixels, are taken at a 5-meter interval. While driving, a 360° LIDAR scanner is continuously scanning the environment. This Velodyne LIDAR scanner, with 32 planes, is seen in figure 1.1b. The scanner is tilted backward, which means it scans a small part of the scene which consists of the road just behind the car, a small part of the scene next to the road and also higher buildings. By continuously scanning a small part of the scene while driving, a high-density point cloud of the whole street can be recorded.



(a) (b) (c)

Figure 1.1: Recording setup of CycloMedia. Recording system DCR10 which records 100 megapixel images (a), a Velodyne HDL-32E LIDAR scanner (b) and the full system assembled on a Ford Fiesta recording car (c)

An important aspect of the CycloMedia data is the highly accurate positioning and calibration of the images. Positioning on GPS alone is unreliable and in urban areas, a GPS signal might be hard to receive. Therefore, the position of the recordings is calculated by combining several sensors like GPS and IMU. Because of this, the relative position accuracy between Cycloramas is within 2 centimeters. Besides that, the images are precisely aligned. The location of the absolute north in the images is known and because of this, the yaw angle of every pixel in the image can be calculated. The vertical direction is also carefully aligned, which means the horizon is in the middle of the image and the $-90°$ and $90°$ pitch angles are known, which means the pitch of every pixel can be calculated. With this information, a vector pointing from the recording location into the real world can be computed for every pixel. This is more thoroughly discussed in section 3.1.2.

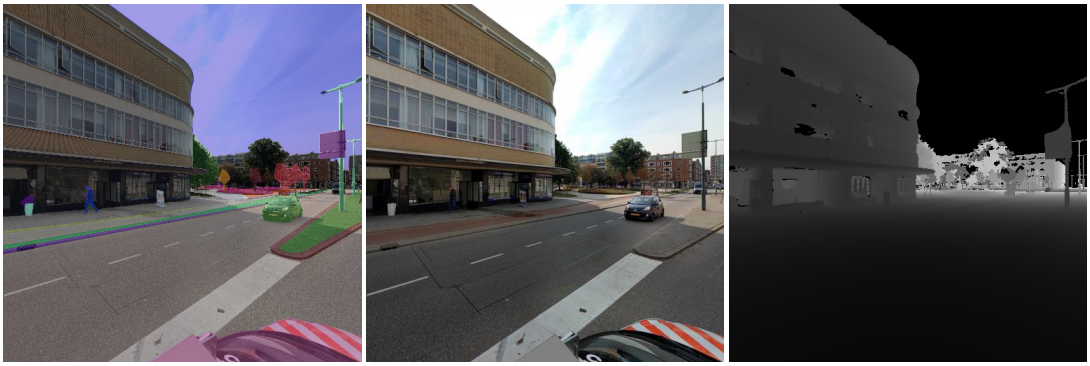


(a) (b)

Figure 1.2: The projection system used to present the panoramic images in this thesis. The cubic projection is a way to visualize the world around the recording locations by dividing it into a cube with 6 tiles. Image (b) is taken from learn.foundry.com

The image data can be shown in different projections. The projection used for this thesis is the cubic projection. This means the world around the camera is visualized by six tiles as if a glass cube is placed around the camera. An example and an explanation of the cubic projection system can be seen in figure 1.2. From now on in this thesis, when showing images with a cubic projection the downwards and upwards facing tile are not shown, as no objects of interest should be present in those tiles. For every pixel in these images, the depth can be calculated by using the 3D mesh which is created from the LIDAR point cloud. This can be calculated by measuring the depth between the camera location and the point where a ray from the camera location intersects with the 3D mesh. This is automatically done in the CycloMedia pipeline. This depth information is stored in an encoded way and can be decoded which results in a depth value for every pixel,

as seen in figure 1.3c. Besides the normal image tiles and the depth images, an annotated dataset became available during this thesis. This is a collection of images tiles on which all objects of interest are precisely annotated by an external company. With more than 90 classes in total, this is a very useful dataset which can be used to train, for instance, segmentation and object detection networks. An example of an annotated image tile can be seen in figure 1.3a.



(a) (b) (c)

Figure 1.3: Examples of data available for this thesis. All images are tiles from the cubic projection of a panoramic image. Image (a) shows a tile from the annotated dataset for object detection and segmentation. Image (b) shows a normal image tile recorded with the DCR10 system. Image (c) shows the depth information based on the LIDAR point clouds.

1.2. Research questions

In section 1.1 the specifications of the available data were described. This data should be used in an efficient way to perform the two main tasks of this thesis. The annotated dataset and LIDAR depth information is mainly suitable for the detection of moving objects. Image data from other points of view is useful for the image inpainting with truthful content. This whole process can be summarized in the following main research question:

- ***Research question:***
  How can the street-view imagery and LIDAR point-clouds be used in an efficient way to generate a pipeline which finds and removes moving objects in street-view imagery and generates a natural inpainting which consists of the actual information behind the moving object?

The task can be divided into two parts. For the first part, the moving objects should be detected. The focus is on a few specific classes of moving objects which are pedestrians, cyclists, and motorized vehicles. Main complications in this part are the large 5-meter baseline between images and the configuration of the LIDAR scanner.

- ***Sub question 1:***
  How can moving objects of interest (motorized vehicles, pedestrians, and cyclists) be effectively detected on street-view imagery using image sequences with a large baseline, LIDAR data and an annotated dataset for object detection and image segmentation?

For the second part of this thesis, the objects should be removed from the image data and an realistic inpainting should be made using the image data from different recording locations. Image inpainting methods utilizing information from different point of views are available, however, they don't always give satisfying results without artifacts. A good solution to this might be to combine the multiple view inpainting techniques with a neural network approach.

- ***Sub question 2:***
  How should the holes, resulting from deleting moving objects from the image data, be inpainted while keeping the correctness of the images and limiting the number of artifacts?

1.3. Contributions

Three main contributions are made in this work:

- **A full pipeline which performs both moving object detection and image inpainting of the holes resulting from the removal of the moving objects on street-view imagery.** The input of the pipeline consists of the 360° street-view image of interest, four other 360° images taken at close range, the recording locations of all images and per-pixel depth information. The output of the pipeline is the street-view image of interest in which the moving objects are replaced by the true background of the moving objects.
- **A moving object detection approach which combines image segmentation and depth information from an 'empty world' LIDAR point cloud.** As image segmentation methods don't distinguish between moving and static images, it has to be combined with a LIDAR-based moving object detection approach. This approach is unique because of the specifications of the LIDAR data and the suggested approach to compare image data with neural network features.
- **A novel method is introduced, using neural networks for image inpainting while utilizing information from other viewpoints.** Building on an existing network, a Generative Adversarial Network is modified to be able to learn to use information from different viewpoints and select information from those views to generate an inpainting which is similar to the true background of the moving object.

1.4. Outline

First, a literature study providing an overview of neural networks and the state-of-the-art literature on (moving) object detection and image inpainting is provided in chapter 2. The higher level considerations of the final pipeline are given in chapter 3. In this chapter, a schematic overview of the pipeline is given and for every part of the pipeline the final working principle is described. In chapter 4, all relevant experiments that led to the final pipeline design are described. This section also shows the results of every separate part of the pipeline. Besides visual results, quantitative results are shown for the parts of the pipeline for which a quantitative measurement is possible. In chapter 5, the final results of the pipeline are discussed and recommendations for future work are given. Finally, in chapter 6 this thesis is summarized and concluded.

2

Literature Study

Moving object detection is a highly active research field. Ego-motion, changing lighting conditions and the complexity of the 'real world' are a few examples of the challenges in this field. Therefore a lot of research is performed on moving object detection. Over the last few years, Convolutional Neural Networks (CNN) became a popular method for object detection. This, however, requires a large amount of precisely annotated data to reach high accuracy. Other conventional methods often require accurate depth information or image data captured at a high frequency.

Removing moving objects from image data results in gaps. A realistic inpainting has to be made to get rid of these gaps. Challenges in the field of image inpainting are largely sized holes and filling the hole with the actual background instead of an estimation based on the surroundings of the gap. Besides the conventional inpainting methods, Generative Adversarial Networks (GANs) is a quickly evolving class of neural networks which can be used for image inpainting.

In both research fields, the recent state-of-the-art performances often makes use of neural network-based methods. As these networks are used a lot in this thesis, this chapter will start with a short introduction to neural networks in section 2.1. In section 2.2, the state-of-the-art in the field of (moving) object detection is described and compared. Finally, methods for image inpainting are discussed in a similar manner in section 2.3

2.1. Introduction to Neural Networks

Traditionally, problems in computer visions involve manually defining a set of rules. In the field of object detection, for instance, detection of vehicles in images includes gathering a feature description of the vehicle. This could be for instance information about the shape, texture and spatial position. These features can then be used to find the vehicles in image data. However, acquiring this feature description is tedious and does not scale well to more classes, which have different features. Neural networks are able to learn a more in-depth feature description of objects with less human effort and scale well to a large number of classes. Neural networks, however, require a substantial amount of 'ground-truth' data and usually require a long training time which could take several weeks. In this section, the basic working principle of neural networks is explained. First, the working principle of the commonly used layers in are explained. Secondly, the architecture of two commonly used deep neural network, on which most state-of-the-art methods described in section 2.2 are built, are explained.

Inspired by biological neural networks, neural networks are a very powerful computational model. Generally, a neural network can be divided into three parts as visualized in figure 2.1. The first layer is the input layer, from which the hidden layers of the network will extract information. In the neural networks described in this thesis, the input of the networks is one or several images. The content of the output layer depends on the goal of the network. For object detection, it could be a simple probability vector of the class of the object in the image. The output of segmentation networks is a separate probability vector for each pixel in the image.

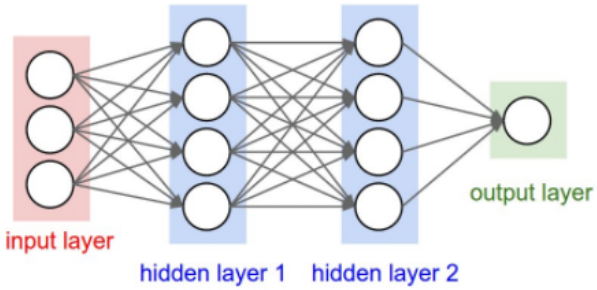


Figure 2.1: The layers of which a neural network consists can be divided into three categories. The input layer is usually an image. The hidden layers perform a large amount of mathematical operations on the input. The final result, the output layer, gives the result depending on the task of the network. In the case of segmentation, for instance, this could be a probability vector describing the class probabilities for every pixel.

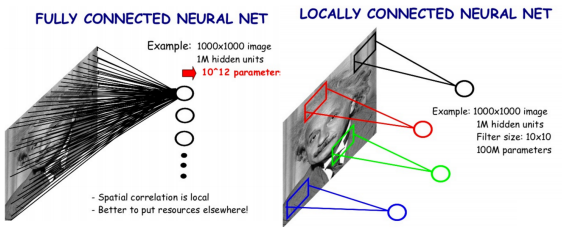


Figure 2.2: Comparison between fully connected (FC) and locally connected / convolutional layers (conv). In a FC layer, each node is a mathematical combination of all the nodes of the previous layer. This results in a huge number of parameters, $N_{\text{parameters}} = N_{\text{nodes layer x}} \times N_{\text{nodes layer x-1}}$. For a conv layer, each node is linked to a limited set of nodes in the previous layer, usually a square patch (kernel) with edge length k. The number of parameters for a convolutional layer is much lower, $N_{\text{parameters}} = N_{\text{nodes layer x}} \times k^2$.

For the hidden part of the neural networks, there are two types of layers which are used, which are fully connected and convolutional (locally connected) layers. The basic principles of those layers are shown in figure 2.2. The nodes in fully connected layers are connected to all nodes in the previous layers. This makes fully connected layers very powerful. This comes, however, at the cost of an enormous number of parameters. In convolutional layers, each node in a hidden layer only takes a small patch of nodes from the previous layers into account. This results in an enormous decrease in parameters and connections, which makes networks much easier and faster to train. The performance of the network does not degrade significantly (Krizhevsky et al. [2]). Because of the low numbers of parameters, a large number of convolutional layers can be stacked at a relatively low computational cost. This led to the popularity of deep convolutional neural networks which are discussed later on in this section [3] [4]. Because convolutional layers are the main building blocks of all neural networks used in this thesis, a more in depth description will be given.

In convolutional layers, a convolution kernel (filter) is passed over the input in a sliding windows approach. The kernel, with size $k \times k$, consists of k^2 weights. The weights of the filter are multiplied with the corresponding values of the input. The results of the k^2 multiplications are added up resulting in one output value for each spatial location on which the kernel is applied. A bias value is added to get the intermediate result as seen in figure 2.3. The intermediate result has a spatial dimension of 3×3, which is smaller than the input matrix of 5×5 because the filter can only be applied in 9 different spatial positions. Usually, the input matrix is padded with a row or a column of zeros on each side. This makes it possible to apply the kernel on the edges of the input, which makes the spatial dimension of the output consistent with the spatial dimensions of the input. A convolutional layer usually consists of a large number of kernels, which are all multiplied with the input in a sliding window approach. This way a lot of different features can be computed on the input. Different kernels with different weights can, for instance, compute edges in different directions or sharpen or blur the input image.

Between the intermediate output and the final output is usually one more step, the activation function. From a mathematical point of view, a convolution is a linear operation. For better training results, it is desirable to add non-linearity to the network. The choice of activation function depends on the network. The three activation functions which are used in the networks used for this thesis are seen in table 2.1. The first one is

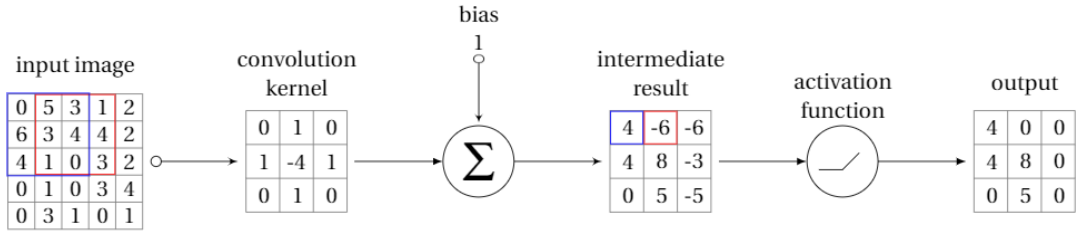
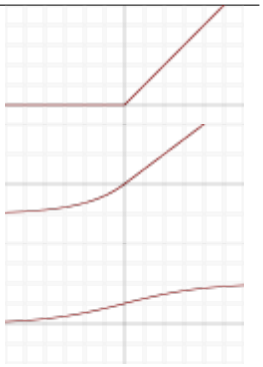


Figure 2.3: Working principle of a convolutional layer. The convolution kernel is iterated over the input image in a sliding window approach. The result of the blue 3×3 square in the input image can be seen in the 1×1 square in the intermediate result. This result is found by multiplying the input image patch with the convolution kernel and then summing the result with the bias: $(0 \times 0 + 1 \times 5 + 0 \times 3 + 1 \times 6 - 4 \times 3 + 1 \times 4 + 0 \times 4 + 1 \times 1 + 0 \times 0) + 1 = 4$. To get the final output, ReLU activation is performed on the intermediate result. The formula for ReLU activation is seen in table 2.1

called Rectified Linear Unit (ReLU), which basically replaces all negative entries with zero. This activation function is also seen in figure 2.3. The second one is the Exponential Linear Unit (ELU) activation function. This activation does not replace all negative values with zeros like ReLU but it limits the range of the negative values by asymptote α with an exponential decrease. Finally, the last one is the Sigmoid activation function. This function translates the output to values within range (0, 1). The Sigmoid function is often used for the final layer of segmentation networks, to create the final probability vector.

Table 2.1: Equation, range and plot of the three activation functions used in the networks for this thesis. The plots in this table are taken from Wikipedia.

Activation function	Equation	Range	Plot
ReLU	$f(x) = \begin{cases} 0 & \text{for } x < 0 \\ x & \text{for } x \geq 0 \end{cases}$	$[0, \infty)$	
ELU	$f(x) = \begin{cases} \alpha(e^x - 1) & \text{for } x < 0 \\ x & \text{for } x \geq 0 \end{cases}$	$(-\alpha, \infty)$	
Sigmoid	$f(x) = \sigma(x) = \frac{1}{1+e^{-x}}$	$(0, 1)$	

The output of the network for a given input image changes when the weights and biases are different. During the training period, the network tries to find the weights which give the best output results. This is done by giving some input with a known desired output (ground-truth) and comparing the output of the network with the desired output. The quality of the output is measured with a loss function. The loss functions used in this thesis are the L1 loss, which is simply the absolute error, and the cross-entropy loss. The equation, range, and convergence measure of these functions are shown in table 2.2. Based on the loss the weights and biases in the network are updated. This process is called back-propagation. During the training of a network, these weights are continuously updated over thousands of iterations, until the results are satisfying. Once the training is finished, the final weight values are stored and the network can be used in the pipeline.

Table 2.2: Equation, range and convergence measure for the loss functions which are used to train the networks considered in this thesis. Here y is the 'ground truth' value and p is the prediction by the network. For the cross entropy, the loss is a sum of the losses of all separate classes, c. In the case of segmentation, the label y is a binary value and prediction p is a value between 0 and 1. These losses are calculated for every pixel, and the mean value is computed.

Name	Equation	Range	Converges to
L1 loss	$S = \lvert y - p \rvert$	$[0, \infty)$	0
Cross-entropy	$S = -\sum_{c=1}^{n} y_c \log(p_c)$	$[0, \infty)$	0

As mentioned above, the low number of parameters of convolutional layers and the increase of the available computational power led to the introduction of Deep Convolutional Neural Networks. These networks consist of a large amount of stacked convolutional layers. The network of Simonyan and Zisserman [3], VGG, can be considered one of the first deep convolutional networks. To be able to increase the number of layers, they decreased the kernel size of the convolutional layers. The disadvantage of lowering the kernel size is a lower receptive field. In a convolutional layer with a kernel size of 7×7 each node considers a larger area of the input image compared to a 3×3 kernel. However, Simonyan and Zisserman argue that a single convolutional layer with a 7×7 kernel can be replaced by three layers with 3×3 kernels. As the last layer with a kernel size of 3×3 still considers all pixel values which are seen by a single layer with a kernel size of 7×7, this does not lead to a decrease in receptive field. Besides that three layers with a 3×3 kernel shows an improved performance compared to one layer with a 7×7 kernel[3]. Intuitively you would expect that increasing the numbers of layers comes at the cost of an increased computational cost, however, this is not the case. One layer with a 7×7 kernel has $7^2C^2 = 49C^2$ parameters, with C being the number of kernels in the convolutional layer. Stacking three convolutional layers with a 3×3 kernel size leads to $3 \times 3^2C^2 = 27C^2$ parameters. This means stacking several convolutional layers with a lower kernel size is not only better for performance, it also is computationally cheaper. By stacking a total of 19 layers, the network of Simonyan and Zisserman outperformed all other networks at the time of publication, with the first place in localization and the second place in classification in the important ImageNet Challenge 2014. Many newer neural networks are still based on (a part of) the architecture of VGG.

Another network architecture on which a lot of the current object detection and segmentation networks are based is ResNet, published by Kaiming et al. [4]. They tried to make an even deeper network than VGG but ran into two problems. First of, vanishing gradient. As discussed before, based on the loss of a network iteration the weights in the network have to be updated. This process is called called back-propagation. This is done by sending a signal from the output layer back through the network to update the weights. If the amount of layers increases a lot, the signal is weak by the time it reaches the first layers. This means the first layers don't get updated accordingly and therefore can't be trained well. The second observation by Kaiming et al. is that adding more layers is not always better. Sometimes the optimal training accuracy does not occur at the final hidden layer and the training accuracy decreases by adding more layers.

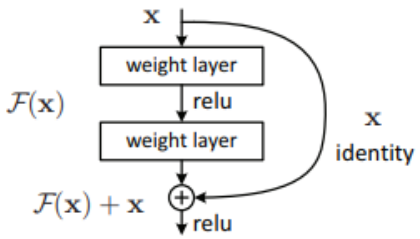


Figure 2.4: A skip connection as proposed by the authors of ResNet[4]. A shortcut is placed after every 2 convolutional (weight) layers. This allows for better training and prevents a decrease in performance which could appear when the peak performance in a network is not in the final layer but at an earlier stage.

In their ResNet paper [4] a simple yet very efficient solution is proposed to solve both these problems. This solution is shown in figure 2.4. A shortcut is added between every two layers. This alleviated the degradation problem during back-propagation because the signal does not have to go through all the layers anymore but can go through the shortcuts. It also fixed the decrease in training accuracy when adding more layers. If an increased amount of layers does not increase the training accuracy, the network can 'skip' these layers. By doing this, they trained networks up to 152 layers without a decrease in performance. The 152 layer architecture showed a significant increase in performance compared to VGG [3].

Neural networks are already showing state-of-the-art performance in various computer-vision problems and the field is still improving at a rapid rate. In the last years, a shift towards deep convolutional networks is visible, where many convolutional layers with a small kernel size are stacked up. With the large numbers of data and the computational power available for this thesis, an approach which includes neural networks has great potential.

2.2. Moving object detection

In the research field of (moving) object detection based on image and/or LIDAR data a wide range of possible methods is available. For the initial broad literature review, the relevant literature is separated into five categories. These categories can be seen in figure 2.3. The main difference between these categories is the specifications of the data which is required. A good way of choosing an appropriate method is to consider which data is available and compare those specifications to the required data of the categories. The available data has been thoroughly discussed in section 1.1. High-density LIDAR point clouds and high-resolution images with very precise positioning. Besides that, a large set of images in which the locations of the objects of interest (cars, pedestrians, cyclists) are precisely annotated is available.

The first two categories which were studied for this thesis are two different types of neural networks. These are object detection networks [5][6][7][8] and segmentation networks [10][11][12][13]. These two categories have very similar requirements. A ground-truth dataset is needed to train the networks. The specifications of the required datasets are different. For object detection, only the coordinates of bounding boxes around an object of interest is required. For segmentation, annotated data on pixel-level is required. These datasets both became available during this thesis, and therefore could be used. To train these networks within a reasonable time a powerful Graphics Processing Unit (GPU) is recommended, which is also available. Segmenting images on a high frame-rate on high-resolution images is not viable, however real-time operation is not required for this thesis.

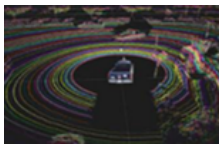
Optical flow [14][15][16][17] is a good approach to moving object detection if the available image data has a high frame-rate, however, the available data is recorded at a 5-meter interval which results in a frame-rate of 5-7 frames per second on a highway and 1-4 frames per second in urban driving. In recent approaches[16][17], depth information is used which improves results on lower frame-rates. Depth information is available through the LIDAR data, however, the moving objects do not appear in the point clouds which makes it unsuitable for optical flow.

The stixels world [18][19][20] is a very efficient way to represent the 3D environment. This representation, in which objects in the real world are represented by vertical bars with a height corresponding with the height of the object, can be used in several ways. It can be used to efficiently find Regions of Interest (ROI) in an image [18] and moving objects can also be found by tracking clusters of stixels [19]. To get a correct stixel representation, however, depth information of the moving objects in the images has to be available. Next to that, to perform good moving object detection on stixels alone, a high frame-rate is recommended.

Many moving object detection methods use LIDAR [21] [22] [23] [24]. With the dense point clouds recorded by LIDAR, a lot of information about the 3D environment is available. Many LIDAR-based publications focus on approaches to turn these point clouds into a 3D grid [21][22]. They describe the 3D world as a grid of occupied and non-occupied 'building blocks'. Groups of occupied building blocks are grouped into objects, and those objects are tracked over time to determine which objects are moving. Other publications combine LIDAR and camera data [23] [24]. Usually, LIDAR is used to find moving objects and their location and size. Based on the location and size, the object can be found in the image. The image is then used to find the correct class of the object. Because of the LIDAR configuration of the CycloMedia recording unit, which is explained in section 1.1, those LIDAR publications are not viable for the available LIDAR data for this thesis.

Based on this short summary, object detection and segmentation networks are the most viable categories considering the available data and resources. From those two, segmentation is the preferred approach as segmentation results are more useful for the inpainting part of this thesis. Segmentation gives pixel-level results, while object detection networks return a square bounding box around the object. This means that, when using segmentation, only the moving object has to be removed from the image instead of a full square around the object. This means a smaller area has to be inpainted. Therefore, a more in-depth literature review on segmentation is included in section 2.2.1 The disadvantage of segmentation networks is that they do not distinguish between static and moving objects. Therefore, segmentation has to be combined with another approach to discover which of the segmented objects are moving and which ones are static. This is a LIDAR-based approach, which is explained more thoroughly in section 2.2.2

Table 2.3: The five described categories of moving object detection methods and their viability considering the available data. The top row are the required resources for the specific categories and the bottom rows are the recommended resources. Green color means the resources are available for this thesis, red means the resources are not available. Note that depth information is available, but not for the moving objects.

				
Object detection	**Segmentation**	**Optical flow**	**Stixels**	**LIDAR**
Ground truth	**Ground truth**	**High framerate**	**Depth info**	**LIDAR scanner**
GPU power	**GPU power**	**Depth info**	**High framerate**	**Full view scans**

2.2.1. Image segmentation

Segmentation networks are very powerful tools in machine learning which learn image classification on pixel-level. Segmentation is the preferred type of neural network for this thesis because it can find the exact outline of the moving object which limits the amount of data that has to be removed. The main difference with object detection networks is that the output layer of segmentation networks usually has the same spatial dimensions as the input image. The output size is $h \times w \times c$, with c being the number of classes in the dataset. Within the hidden part of the network, the spatial dimensions are usually first slowly scaled down and later on scaled back up to the original dimension. Because of this, the convolutional layers in the middle part of the network are able to get more global information about the scene in the image. An example of a network architecture like that is visible in image 2.5. This is called an encoder-decoder structure, where the first half of the network is the encoder and the second half is the decoder. In this section, popular networks and the current state-of-the-art are discussed.

The first promising segmentation results with convolutional neural networks is FCN, published by Long et al. [10]. To make sure the network gets a global understanding of the full image, researchers used to utilize fully connected layers in their network. Long et al. realized that by stacking enough layers and scaling down the spatial dimensions of the feature maps, learning a global understanding of an image is also possible by stacking several convolutional layers. In their work, they stacked a number of convolutional layers and combined it with average pooling to scale the spatial dimensions down after every convolutional layer. After that, a combination of the last layers, with low spatial dimensions, is scaled back up to the original input size. This was not done by normal up-sampling but learned with an upscaling convolutional layer (deconvolution). Nowadays, FCN is no longer a state-of-the-art network, but at the time of publication it was a big improve on the state-of-the-art and it led to a lot of other publications on segmentation with convolutional layers.

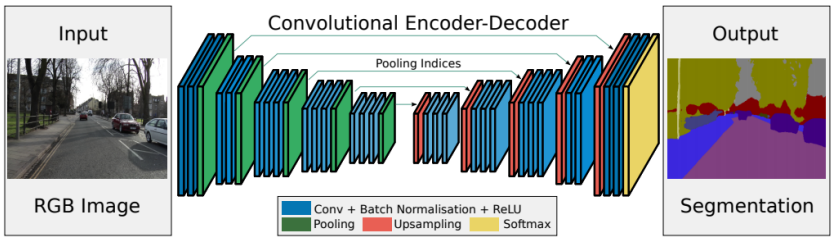


Figure 2.5: Network architecture for SegNet[11]. An encoder-decoder structure with convolutional layers, max pooling down-sampling and up-sampling based on the pooling indices of the max pooling.

The next big improvement on image segmentation is SegNet, published by Badrinarayanan et al. [11]. The results of FCN[10] still lacks details which is, according to Badrinarayanan et al., caused by the very sudden upsampling and the lack of shortcut connections. As a fix for this, they propose an encoder-decoder structure for image segmentation. With this architecture, the spatial dimensions are scaled back up step by step instead of one sudden increase as presented in FCN. A schematic overview of the network architecture can be seen in figure 2.5. The encoder and decoder are both based on the network architecture of VGG-16, as published by Simonyan and Zisserman [3]. As seen in the figure, there are also shortcut connections between the encoder

and the decoder. These are the pooling indices of the max pooling. Max pooling is a way of down-scaling images or feature maps. To reduce the image size with max pooling with a factor of 2 in both dimensions, for every 2×2 square in the original image the maximum value is taken. The index of the maximum value of the 2×2 square is passed on to the decoder, in which it is used to improve the up-sampling. Just like FCN, SegNet was a big improvement on the state-of-the-art when published. SegNet is sometimes still used, because of its simplicity compared to the current state-of-the-art in image segmentation.

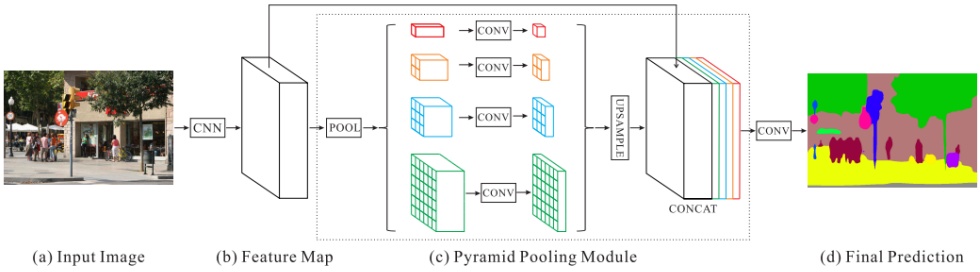


Figure 2.6: Network architecture for PSPNet. After an initial feature map computation with ResNet [4], global information is captured by using max pooling. This information is concatenated to the ResNet feature map, giving the final convolutional layer a bit more context about the full scene.

The two previous models both had a big influence in the field of segmentation and inspired many newer models. Although both improved the state-of-the-art significantly at the time, newer networks show better performance. Two more recent networks with state-of-the-art performance are PSPNet [12] and DeepLab V3 [13]. Both these networks build on earlier networks but also have their own unique contributions. With the Pyramid Scene Parsing Network (PSPNet), Zhao et al. [12] present a unique way of using global information, shown in figure 2.6. First, a regular CNN, ResNet [4] in this case, is used to get a feature map of the image. After that, with average pooling (similar to max pooling, but now the average value of a patch is taken) more global features of the image are found. For the top feature map, red in figure 2.6, max pooling is performed on the full image, resulting in a 1×1 feature map which can be seen as a summary of the scene in the full image. In the other average pooling operations, each part of the output describes a smaller part of the image, which means that as the output size gets bigger (red to green in figure 2.6) the information is getting less global and more local. These global and local features are up-sampled to the spatial dimensions of the input feature map and then concatenated to the input feature map. Based on this original feature map with the global information concatenated to it, the last convolutional layer computes the final prediction. This approach results in a prediction in which objects are partly classified based on the context of the full image.

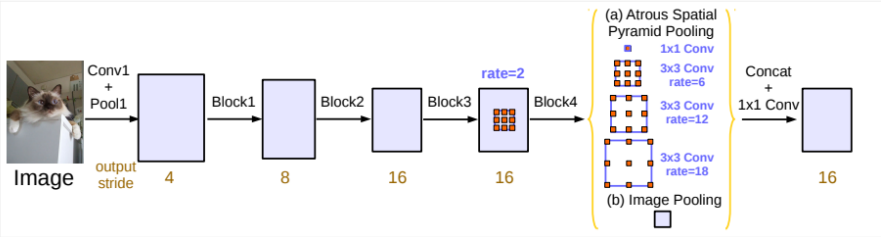


Figure 2.7: Network architecture for DeepLab V3. After an initial feature map is computed (best results where obtained by using ResNet [4]), atrous spatial pyramid pooling is used to get a more global feature map of the scene. Based on this information, the final predictions are done.

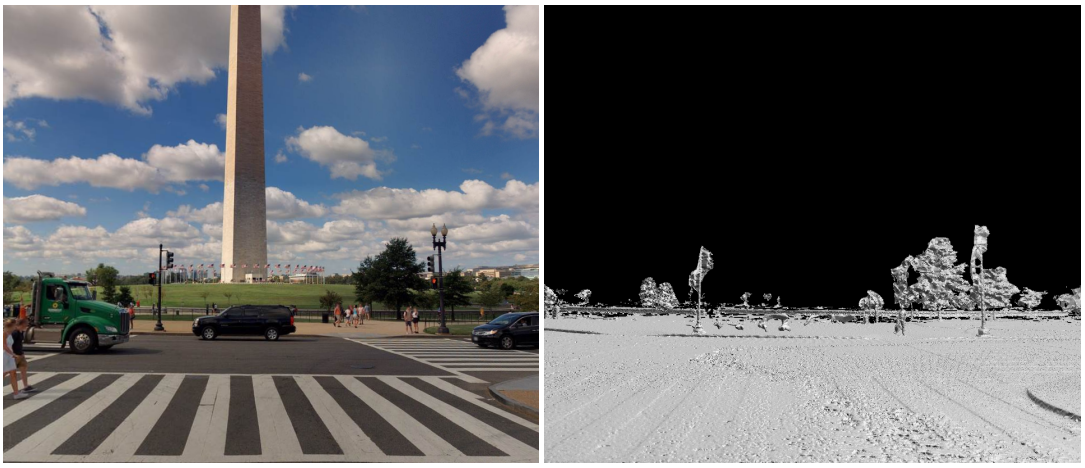
The last model presented in this section is DeepLab V3, a more recent model presented by Chen et al. [13]. In this model, in contrast to the average pooling in PSPNet[12], atrous convolution is used to extract global information. Atrous convolution, also known as dilated convolution in other literature, is a type of convolution with a different type of kernel. A 3×3 kernel is used, but instead of considering the 3×3 closest nodes of the previous feature map, some nodes are skipped as seen in figure 2.7. This way, with a cheap 3×3 kernel size it is still possible to capture global information. By increasing the rate, more global information of

the image or feature map is obtained. With convolution at different rates, both global and local information is obtained. This information is concatenated in one feature map with both global and local information, which is used to get the final pixel-level prediction. This is a good way to get a good receptive field without scaling the spatial dimensions of the feature map down too far or making the network too computationally heavy.

The state-of-the-art in segmentation has improved a lot over the past few years. The best performing model is currently DeepLab V3, which is described above. Segmentation networks are usually compared by their mean IoU score on public datasets. For our case, the most interesting public datasets are the CityScapes dataset[38] and the Mapillary Vistas dataset[39]. Segmentation models are a very powerful tool and pixel level classification is more useful than normal object detection networks for removing occlusions. Segmentation models, have to be trained on a big carefully annotated dataset, which came available during this thesis.

2.2.2. Utilization of CycloMedia LIDAR data

While segmentation networks can find objects of interest at a high accuracy, it does not distinguish between static and moving objects. A secondary, LIDAR based, approach is needed to determine which objects are moving. As discussed before, the configuration of the LIDAR scanners on the CycloMedia recording vehicles is different from the configuration seen in most LIDAR based object detection publications. In these publications, the scanner is usually mounted on top of the car with a 360° view of the surroundings for every scan. With this information, each scan gives a full 360° view of the surrounding, and by computing the difference between different scans the moving objects can be found. The LIDAR scanners on the CycloMedia recording vehicles are tilted backward. This means that every scan, a small part of the road behind the car, a part of the scene next to the car and also higher buildings are captured. With this configuration, a small part of the scene is captured with a high density every scan and by continuously scanning while driving a high-density point cloud of the whole street is captured over time.



(a) (b)

Figure 2.8: Visualization of the inconsistency between image data (left) and the corresponding mesh based on LIDAR data. As seen in the images, the moving objects in the image (a) are not visible in the LIDAR mesh (b). Note that the objects at a large distance are also not visible. This is due to the range of the LIDAR scanner.

Because of the limited field of view, most state-of-the-art LIDAR based object detection publications are not applicable for this thesis. Another problem of this LIDAR configuration is that moving objects are usually not recorded with the LIDAR scanner. Because of the small field of view, only vehicles directly behind or next to the recording car are recorded. Vehicles outside of that range don't get recorded at all. Even when vehicles are in the field of view of the LIDAR scanners they are usually only there for a short time, which leads to a sparse point cloud. Because the rest of the scene is recorded in a high density, these sparse parts in the point clouds are considered as errors and therefore are filtered out of the point clouds in the LIDAR processing pipeline. This leads to point clouds in which only the static objects are visible, an 'empty world' without moving vehicles. From the LIDAR point clouds a mesh is made. This is an approach in which points from

LIDAR point clouds are connected in triangles resulting in a more 'solid' world. An example of a mesh, in which moving objects that appear in the image data are not visible can be seen in figure 2.8. Besides normal LIDAR based object detection methods, this also limits the use of other depth-dependent methods like optical flow and stixels. It also leads to an inconsistency between image data and LIDAR data. Both data types are recorded at the same time in the same scene but one of them does not show moving objects. A good approach for this research would be to try and find these inconsistencies and therefore also find the moving objects.

To find these inconsistencies between image and LIDAR data, the two data types have to be linked. One way to do this is by matching images taken at different locations, based on the depth information. With the highly accurate positioning and the depth information available at CycloMedia, corresponding pixels between two images can be found. A method to match omnidirectional images is presented by Marković et al. [25]. As omnidirectional images describe the full view around the recording location, every pixel in one image can always be translated to the images from the other location, unless there is an object occluding the part of the scene described with the pixel. This is visualized in figure 2.9. An omnidirectional image could be projected inside a unit sphere around a recording location. Each pixel on the first sphere can be matched with a corresponding arc on the other sphere. The location of the corresponding pixel on the arc depends on the depth of the object described by the pixel. With the LIDAR point clouds, this depth information is known. Therefore, because of the depth information and accurate positioning, every pixel in one image can be matched with the corresponding pixel in an image from a different location just by geometric calculations. This approach can be used to match images, based on LIDAR data. At places where this matching is not correct, there is an inconsistency between LIDAR and image data, which is possibly a moving object. More about the implementation of this principle can be found in section 3.1.2.

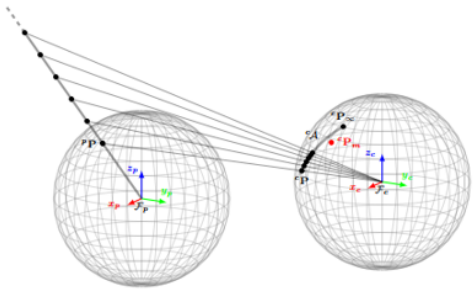


Figure 2.9: Pixel matching method for calibrated omnidirectional images. Because of the full field of view around the images, every pixel in one image can be matched with a pixel on an arc on the other image, unless the object described by the pixel is occluded by another object in the other image. The location of the pixel on the arc is dependent on the depth.

Data from the CycloMedia LIDAR scanners is recorded in a different way compared to LIDAR data used for object detection publications. This results in moving objects not being visible in the point cloud data. This is a big limitation as the categories optical flow, stixels, and most LIDAR based methods are not viable with these point clouds. Even though the LIDAR configuration limits all those methods, it also results in a nice possibility. The moving objects are an inconsistency between image and LIDAR data. By combining these two types of data, moving objects can be found.

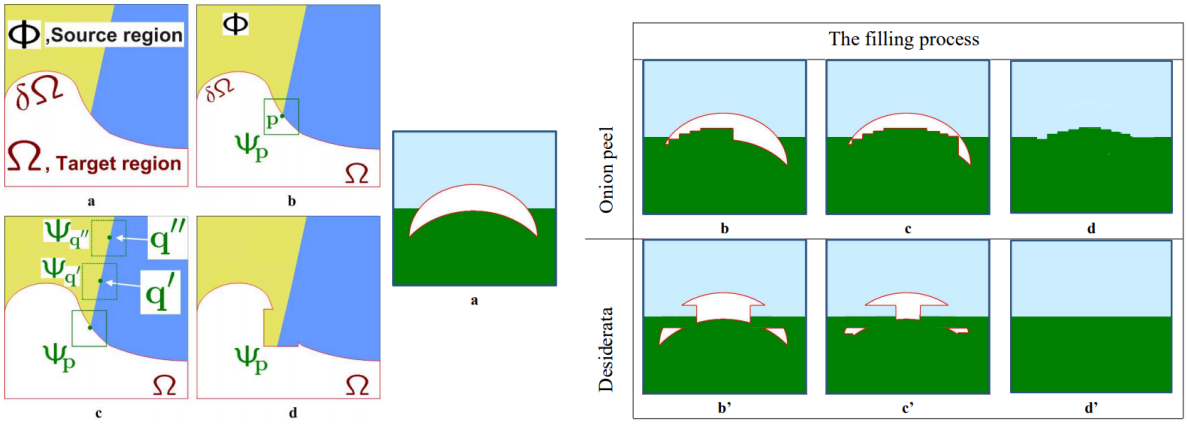
2.3. Image inpainting

Finding moving objects in image data is the first step of this thesis. The second step is to remove the moving objects from the image and fill the hole that arises. This can be done with information from the surroundings of the hole and information available from other viewpoints or image data from different years. The approach of filling holes in images is usually referred to as image inpainting. For the image inpainting of the CycloMedia data, there are two requirements. First, the data should be realistic. It should not be clearly visible that a part of an image has been removed and filled up with other data. Secondly, the hole should be filled with the correct data. This means that the method used to fill in the holes should make use of context from other viewpoints instead of just an estimation of the content behind the moving vehicle. The literature on image inpainting can be roughly divided into three categories. In this subsection, these categories are discussed

first. The quality of the inpaintings generated by these categories is compared with the requirements for the inpainting. Finally, this section will end with a proposition of a new method which is a combination of two of the described categories.

2.3.1. Exemplar-based inpainting

The first category, exemplar-based inpainting, is the most basic one. Both other categories are partly based on this approach. The basic process of exemplar-based inpainting is shown in figure 2.10a. In this figure, patch ψ_p around pixel p on border $\delta\Omega$ is filled with the most comparable patch. By repeating this process, slowly the hole is filled until the final result is found.



(a) (b)

Figure 2.10: Exemplar-based inpainting example (a) and comparison between onion-peel and desirable method. (b). Image (a) shows how a pixel on the boundary (ψ_p) is selected and a patch which is similar to the patch around pixel ψ_p is found and copied to the boundary. Figure (b) shows how prioritizing known structures instead of working from the outside towards the inside leads to better results.

The most crucial part of this process is the order in which the pixels are selected for inpainting. This was brought to attention by Criminisi et al. [26], one of the first papers on exemplar-based inpainting. Simply inpainting from the edges to the center, in this thesis referred to as the *onion-peel* method does not always give a good result. They propose a method which first makes an accurate inpainting of the known structures and then fills in the other parts of the gap. This is done by computing a confidence factor of all the pixels on the boundary. Pixels with known structures are prioritized because they are more likely to be correctly inpainted. After finding the pixel with the highest priority, a patch in (a part of) the rest of the image which describes the area around pixel p best is searched. This patch is used to fill in the unknown parts of the patch around pixel p. After this, the pixel priorities are updated and the process is repeated for the new pixel with the highest priority.

The search for a comparable image patch is the most time-consuming part of the above algorithm. Barnes et al. [27] realizes that for neighboring pixels at the boundary of the gap, it is likely that good corresponding patches are close to each other. To speed up the process, they first do an initialization, in which a lot of pixels at the boundary are compared with patches in the rest of the image. When one of the pixels at the boundary of the hole is decently matched with an image patch in the rest of the image, other boundary pixels around the considered pixel are also matched with patches nearby the correct patch. After this, the region around the correct patch is searched for even better patches. These three steps are illustrated in figure 2.11. With this method, called PatchMatch, an interactive tool could be made which makes almost real-time inpaintings of user-initialized gaps.

With these two papers, a basic explanation of possibilities and bottlenecks of exemplar-based inpainting is given. A more in-depth review and a comparison of exemplar-based inpainting methods are presented by Buyssens et al. [28]. Exemplar-based inpainting is widely used in, for instance, PhotoShop and many of the papers in the next two sections are based on exemplar-based inpainting.

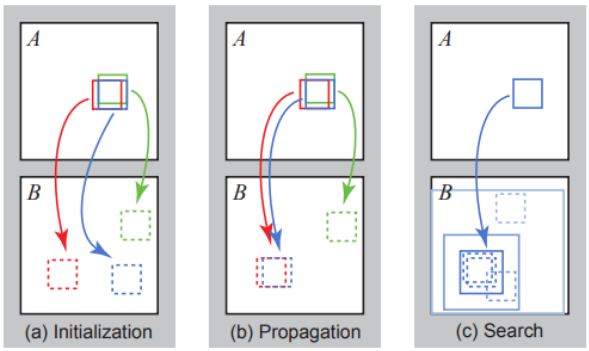


Figure 2.11: The algorithm proposed by PatchMatch [27] to speed up and improve image inpainting. First patches are randomly selected. When a boundary patch is successfully matched with a patch from the outside, the search for matching patches for neighbouring boundary pixels is performed in the same region.

2.3.2. Multiple view inpainting

Previous methods try to make a realistic inpainting by propagating known structures at the boundary into the hole. In some cases this is sufficient, but with larger holes and the complexity of street-view imagery, some more context is necessary. Therefore some publications utilize information from other point of views. Many state-of-the-art publications perform inpainting with stereo images [29] which means the baseline is very low. However, because the large 5-meter baseline between the images at CycloMedia, the publications which use a bigger baseline are more interesting for this thesis.

Within the publications about multiple view inpainting with a big baseline, there are two different approaches. Methods that use a copy-and-warp approach to inpaint the holes and methods that use an exemplar-based approach. A good example of a copy-and-warp approach is the work of Flores and Belongie [30]. In this work, pedestrians are removed from street view imagery. Homography between two views is computed using SIFT and RANSAC. Using this, the bounding boxes around pedestrians are warped to images from other views. If the bounding box in the other view does not contain any pedestrians, this bounding box is warped and copied to the original view to replace the pedestrian. This method has its limitations. Only two images are used and therefore it often occurs that the area of interest is also covered by a pedestrian in the other image. To work around this problem, more than two views should be used or a regular inpainting approach should be used for the area that is not visible in any other view.

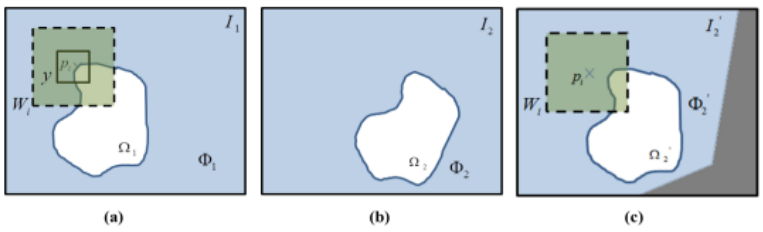


Figure 2.12: inpainting based on multiple views. The white area in image (a) is the area that has to be inpainted. Image (b) is an image of the same scene from a different angle. Image (c) is the result of warping image (b) to fit image (a). By doing this, it becomes very clear that some parts of the occlusion of image (a) can simply be taken from image (b), and for other parts a different view or approach has to be used.

Thaskani et al. [31] use an exemplar-based approach to perform inpainting based on multiple views. To do so, they first rotate and warp images taken from different point of views to get aligned views. By doing this, it is easy to see which part of the occluded area is not occluded in the other picture. This can be seen in figure 2.12. Image (a) is the area that has to be inpainted. Image (b) is a different view of the same scene. Reprojecting this image to the point of view of image (a) results in image (c). Here it can clearly be seen that the area around pixel p_i, which is on the boundary of the hole in image (a), is not occluded in image (c). Thonat et al. [32] consider the fact that vehicles are often close to a building, which makes the actual content not visible in other views. In their approach, first, the areas that can be seen from other views are inpainted. After

that, for the parts of the occlusions which are not visible in all other views a PatchMatch [27] based approach is used, selecting patches from not only the considered view but also other views.

2.3.3. Generative Adversarial Networks

The last analyzed field in inpainting is a very new and rapidly growing one, Generative Adversarial Networks (GANs). These neural networks consist of two parts, a generator and a discriminator. The general idea of GANs is that the generator takes an input (often Gaussian noise) and the output of the network could be for instance an image of a face. The generated image of the face is fed to the discriminator, together with a real image of a face. The discriminator tries to distinguish between real and fake and then gives feedback to the generator. The discriminator could be considered a sort of 'learning loss function'. Both the generator and discriminator improve over time during training and therefore also improve each-other.

In the case of inpainting, the input to the generator network is an image with an occluded area and usually a binary mask which marks the occluded area. The desired output is the inpainted version of the input image. During training, the output of the generator is fed to the discriminator. The discriminator compares the inpainted image to a ground-truth image. The discriminator learns to measure the quality of the inpainting and gives feedback to the generator. The generator and discriminator play this min-max game for thousands of iterations and by doing that improve each-other. With this approach, GANs have shown to be a good approach to image inpainting with state-of-the-art results.

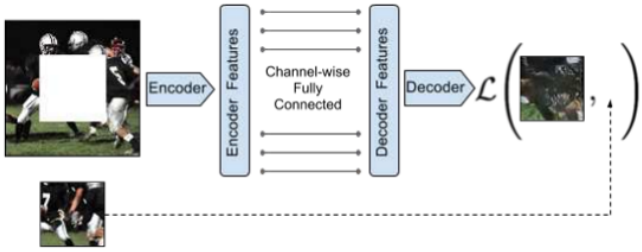


Figure 2.13: Schematic overview of Context Encoders [33]. The image with an occluded area goes through an encoder-decoder network and the output is compared with the ground truth. This network can be trained to learn exemplar-based inpainting by interchanging patches in the fully-connected layer.

The first successful attempt to use neural networks for inpainting is Context Encoders, published by Pathak et al. [33]. In this paper, an auto-encoder is used for image inpainting. This is comparable to the GANs which became popular at a later stage, but the comparison of the image with the ground-truth is done by a loss function. GANs replace this loss function with a discriminator network which can be seen as a 'learning' loss function. A simplification of the layout is shown in figure 2.13. The input image is scaled down with a convolutional encoder, scaling the spatial dimension from 227 x 227 down to 6 x 6. On that final feature map, a fully connected layer is applied which means the different patches from all over the image can be interchanged because, as explained before, in a fully connected layer each node is connected to every node of the previous layer. This approach, which is used by most inpainting GANs, is comparable to the process of scanning the image for similar patches in exemplar-based methods. After the fully connected layer, the spatial dimensions of the feature map are scaled up back to the original image size in five steps, using convolutional layers. Finally, the inpainted part of the image is compared to the ground truth. The more recent inpainting GANs which are described later in this section are based on this network.

One of the many improvements on Context Encoders is presented by Iizuka et al. [34]. The 'completion network', which is seen in figure 2.14, is comparable to the encoder-decoder structure of Pathak et al. [33]. The main difference is the step in which the patches are distributed across the image. In Context Encoders, a fully connected layer is used for this purpose. Iizuka et al. [34] propose a computationally more efficient method by using dilated convolutions. Dilated convolution, as explained in section 2.1 and in Chen et al. [13], are convolutional layers in which the receptive field of kernels is widened by skipping pixels. The other main change presented in this work is the use of a discriminator. Pathak et al. [33] compute a simple L2 loss over

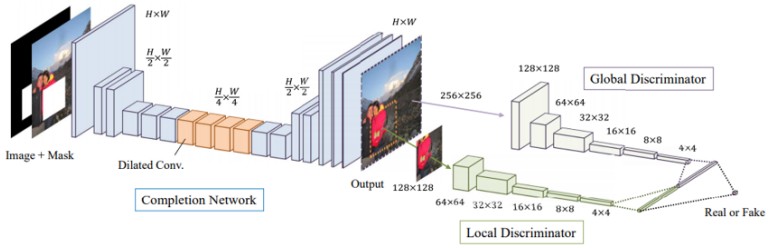


Figure 2.14: Network architecture of Iizuka et al. [34]. The first part, completion network, is comparable to Pathak et al. [33], but the fully-connected layer is replaced with dilated convolutional layers. The output of the network goes into the global discriminator (whole output) and local discriminator (inpainted part), which learn to distinguish inpainted images from real images and therefore improve the generator.

the inpainted patch. Iizuka et al. [34] replace this with two discriminators which makes this network a real GAN. The local discriminator only measures the inpainted part and makes sure the inpainted part consists of the right content. The global discriminator considers the whole image and measures if the inpainted part fits nicely into the image and if the other parts of the image remained the same.

Another improvement on Context Encoders [33] which is done in several publications is the use of a second generator network. The aim of the network is to refine the inpainting result. The output of inpainting networks is often blurry without fine details. This output can be used as a first prediction, and fed into a second network which uses the initial prediction to find better patches in the other parts of the image and use those patches to fill in the details. Zhao et al [35] use this approach. In their network, the full output image is fed into the second network which is considered a 'deblurring-denoising network'. The output of the second network shows improved detail. Song et al. [36] show a similar approach in which the first part of the second network uses the VGG-19 architecture, which is known to give a very good feature description of an image. On this VGG features, patches are exchanged and the output of the network shows improved detail. Both the output of the first and the second network are used as an input for the discriminator. This method shows good inpainting results on 512×512 images, which is desirable for this thesis.

It is clear that most current state-of-the-art image inpainting GANs are just a specific improvement on Context Encoders, the first publication on inpainting with neural networks. Some of these improvements are dilated convolutions instead of fully connected layers [34], the use of a local and global discriminator for a correct inpainting and global consistency [34] [35] [36], and using a secondary network to improve the detail in the output image [35] [36]. Perhaps the best results can be achieved by combining those improvements into one network. This has been done by Yu et al. [37]. Their work shows some good results achieved with a network that has all three above-mentioned improvements on Context Encoders.

Generative Adversarial Networks are a relatively new field in image inpainting and already shows state-of-the-art results. With the millions of parameters, neural networks are able to learn more complicated relationships and transferring operations compared to conventional approaches. For these networks to perform well, however, a good dataset with ground-truth information is necessary for training the network.

2.3.4. Comparison

For the inpainting of the deleted moving objects in this thesis, there are two main criteria. First, the inpainting has to look natural. No big artifacts should be visible in the final result. The second criterion is that the inpainting should consist of the correct content. With the images from other viewpoints, information is available about the actual background of a moving object. This should be utilized to get an inpainting with the actual content instead of an estimation. The three previously discussed categories are analyzed based on those two criteria. In table 2.4, a compact overview of this analysis is given.

Exemplar-based methods are the most conventional approach to inpainting. The best results are achieved by first propagating structures further into the hole and after that fill the other parts of the hole. However, without accurate depth information, it is hard to say how far these known structures should propagate into

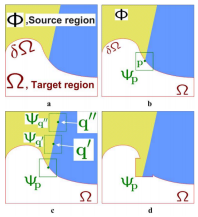
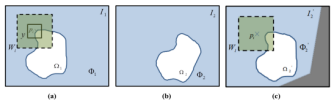
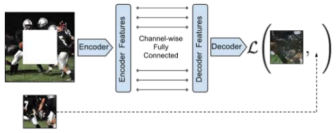
the hole. Therefore the content is often not correct. By simply copying and pasting patches, it is not always possible to generate a realistic output without artifacts. Therefore, the natural look of the output is mediocre and the true content is not visible as it is just a prediction based on the surroundings.

For inpainting from different views, there are two different approaches. The exemplar-based approach is similar to the previous category, but the search range is extended to different views which makes it possible to select patches of the true content instead of just the surroundings. The other approach is to copy the full background of the occluded area from the other view, rotate and warp it to the right point of view and paste it into the hole. This still leads to artifacts, but the inpainting consist of the actual content instead of a prediction.

Generative Adversarial Networks are a powerful tool for inpainting. The approach is comparable to exemplar-based inpainting but the selection of patches is learned with millions of parameters. If a good dataset is available, these neural networks can be trained to generate inpaintings of images at a higher quality than conventional methods. There is, however, no neural network that uses information from different views for the task of image inpainting.

As seen in table 2.4, none of the three categories achieve the full score for both requirements. A possible solution is to combine of multi-view inpainting and Generative Adversarial Networks. By giving the neural network information from other points of view, it might be able to learn to select information from those other views and use this to get a true inpainting of the gap. To the best of my knowledge, no literature is available on this subject and therefore it could be a promising direction for research.

Table 2.4: Comparison between the three discussed image inpainting categories. The results of the categories are evaluated based on the two requirements for the inpainting task, a natural look without artifacts and the inpainting of the actual content instead of a prediction.

	 **Exemplar-based**	 **Multi-view**	 **Generative Adversarial Networks**
Natural	+/-	+/-	+
Content	-	+	-

In this section, the current state-of-the-art on moving object detection and image inpainting is discussed. First, a short introduction to neural networks is given, as they are used in state-of-the-art publications in both fields. For moving object detection, a combination of image segmentation and a LIDAR-based approach is suggested. The required dataset for image segmentation is available, but the second LIDAR-based approach is necessary to distinguish between static and moving vehicles. For image inpainting, there is also no category which meets both requirements. Therefore a combination between multi-view image inpainting and Generative Adversarial Networks is suggested. This is currently a gap in the literature. In the next section, the methodology for this thesis is described.

3

Methodology

For this thesis, a full pipeline has been designed which can fully automatically find moving objects, remove them and make an inpainting of the holes. In this chapter, the final implementation of the different parts of the pipeline discussed without going into too much implementation details. The experiments and validation tests that led to the final implementation and the achieved results are shown in section 4. A simplified overview of the pipeline can be seen in figure 3.1. In this figure, a clear distinction is made between the moving object detection and the image inpainting part of this thesis. In this chapter, the higher level considerations of all the parts of the pipeline are explained by discussing the separate blocks in the scheme. First in section 3.1 the moving object detection part of the pipeline is explained. After that, in section 3.2 the image inpainting part of the pipeline is explained.

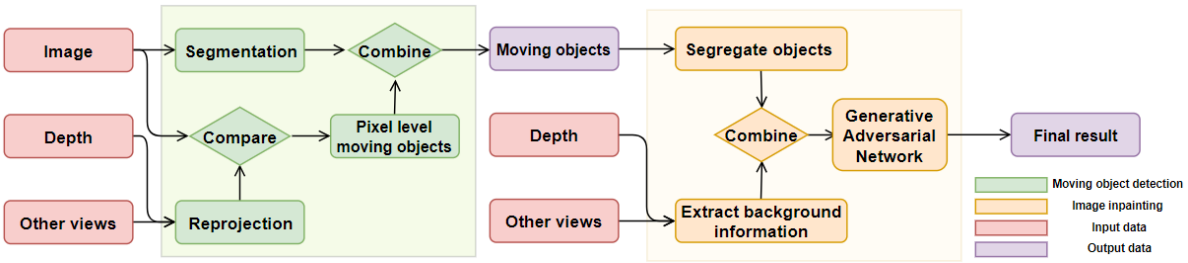


Figure 3.1: Simplified overview of the full pipeline. The input data consists of an image, image data from near point-of-views and depth information. The first part of the pipeline is the moving object detection part which outputs masks of the locations of the moving objects. The second part is the image inpainting part which uses the masks and information from other point of views to make an inpainting with a Generative Adversarial Network.

3.1. Moving object detection

The moving object detection part of the pipeline is, as explained in section 2, a combination of segmentation and a LIDAR-based approach. The results of both approaches are combined at the end of the pipeline as seen in figure 3.1. The segmentation part is done by running a trained segmentation model on the original image in which we want to find the moving objects. The LIDAR-based part is a comparison between the original input image and a re-projection of four images which are the previous two and the next two images of the image sequence which means they are all taken within roughly 10 meters of the original image. In this section, each block of the moving object detection pipeline is discussed separately.

3.1.1. Image segmentation

In the image segmentation part of the pipeline, all objects of interest should be detected. These are cyclists, pedestrians, cars and other kinds of motorized vehicles. This can be done by training a segmentation network on the annotated CycloMedia image dataset. This dataset includes roughly 4000 images which are carefully

annotated by professionals. Only objects within a range of 40 meters are annotated. All kinds of objects, divided into more than 90 classes, are annotated. The majority of these classes are not relevant for this thesis, and therefore the dataset is altered first. In this section the way in which this dataset is alternated is described. Before that, first the chosen segmentation network and the changes that have been made to this segmentation network are described.

Several things should be considered when picking a segmentation network for this thesis. The main consideration is the trade-off between accuracy and speed. As the network will be used intensively when removing objects from images, a low computational cost is preferred to speed up the pipeline. Simplicity is also an important factor. Because the network is also used for feature extraction (explained in section 3.1.3) it is desirable that the network is a simple encoder-decoder structure instead of the current state-of-the-art networks that use the computationally heavy ResNet architecture [4] or use several branches like Mask R-CNN, the latest state-of-the-art network by He et al. [9]. A network with an encoder-decoder structure which achieved state-of-the-art performance when published is SegNet, published by Badrinarayanan et al. [11]. The architecture of this network is discussed in section 2.2.1 and an image of the network architecture is shown in figure 2.5. Even though the network was outperforming the state-of-the-art when published, some of the implementation details are slightly outdated. Therefore, SegNet was slightly altered for this thesis with two main changes:

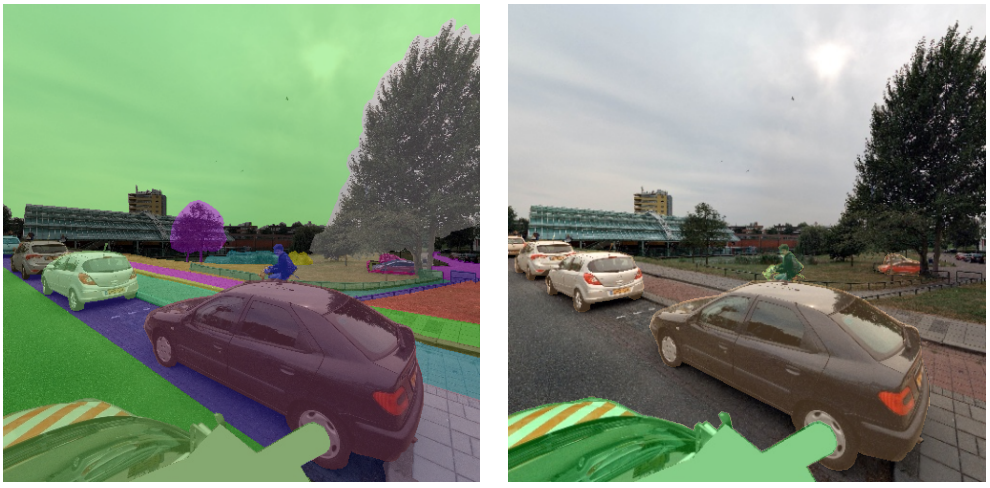
- **Replacing the 7×7 convolutional layers.** SegNet uses convolutional layers with a kernel size of 7×7. This results in a lot of parameters and can be improved as proven in the publication of Simonyan et al. [3]. In their VGG network, they replace 7×7 layers by three consecutive 3×3 layers. This combination of the three 3×3 layers results in the same receptive field as one 7×7 layer and almost halves the number of parameters. According to Simonyan et al. [3] it also leads to improved result. Therefore, all 7×7 layers from the original SegNet publication are replaced by three consecutive 3×3 layers.

- **Removing the shortcuts of the max-pooling indices.** In SegNet, the indices of the max-pooling layers in the encoder are used for the upscaling of the feature maps in the decoder. In the current state-of-the-art networks, this approach is no longer used. Instead, upscaling convolutional layers or simple bi-linear interpolation is used to scale up the feature maps. As bi-linear interpolation is a very cheap process and it is desired that the network has a low computational cost, bi-linear interpolation is chosen to scale up the feature maps in the decoder part of the network.

With those changes, SegNet is updated to be more compliant with the current state-of-the-art while still keeping the desired simple encoder-decoder structure. The network is considerably faster than the original implementation and should give comparable or better results.

To train the network, a ground-truth dataset is available. This dataset became available during this thesis and could be used to train segmentation. For this dataset, a group of trained annotators drew the outlines of objects in image tiles and assigned a class label to those objects. This resulted in a dataset of 4000 annotated square image tiles. This dataset consists of a total of 96 different classes. This means the dataset includes a lot of fine details like CCTV cameras and road markings which are not interesting for this thesis. Next to that it includes separate instances of all these objects and masks overlap which means several classes can be assigned to one pixel. As a relatively straightforward network is used for this thesis and only a few types of classes are of interest, it is better to simplify this dataset. Therefore, the dataset is divided into five different classes. The first class is the CycloMedia recording car. This class is separated from the other cars as they appear significantly different in the images compared to other cars. The recording car should not be removed from the image data which makes as clear distinction between the CycloMedia car and other cars useful. The second class is the vehicle class which includes all kinds of motorized vehicles like cars and trucks but also caravans and trailers which usually appear together with motorized vehicles. The third class is the bicycle class which also includes other similarly shaped vehicles like motors. The fourth class is pedestrians which also includes riders of bikes and other vehicles. Lastly, every class which is not interesting for this thesis is combined in the fifth class. A full list of the classes of potentially moving objects and the division made for the new dataset is can be seen in table 3.1. An example image of the simplification can be seen in figure 3.2.

The distribution of the classes in this dataset is highly unequal. This could be a problem during the training phase of the network. The loss which is used for SegNet is the Cross Entropy loss, as explained earlier and



(a) (b)

Figure 3.2: Example of the simplified dataset which is used for training the segmentation network. The old dataset (a) includes every separate instance of the objects and has a total of 96 classes. The new dataset (b) is limited to only 5 classes without separate instances. Note that the fifth class, objects not of interest, is not color coded in this image.

Recording car	**Pedestrians**	**Bicycles**	**Motorized vehicles**	
Car mount	Person | Individual	Bicycle | Single	Car | Single	Boat | Group
Ego vehicle	Person | Group	Bicycle | Group	Car | Group	Other vehicle | Single
	Rider | Bicyclist	Motorcycle | Single	Trailer	Caravan
	Rider | Motorcycler	Motorcycle | Group	Truck	Bus | Single
			Wheeled | Slow	Bus | Group
			Boat | Single	Vehicle on rails

Table 3.1: The division of the 96 classes of the CycloMedia dataset into the dataset with 5 classes which is used for this thesis. Classes which are not mentioned in this table are all merged in the fifth class. This class consists of all kinds of things which you can see in a general road scene like nature, buildings, sky, road markings and traffic attributes.

shown in table 2.2. When calculating the loss of a training iteration, the separate loss for each pixel is calculated and the mean value over all pixels is computed. If a high percentage of the data consists of one class, the loss is mainly focused on that one specific class. This means that the network prioritizes this class and focuses on recognizing that class first during training. The fifth class in the created dataset, with all objects that are not interesting, is the biggest class by a big margin. This means that the network focuses on learning information which is not of interest for this thesis. To fix this problem, the loss is weighted. To do so we calculate the distribution of the classes in the dataset. The distribution of the classes of the full training dataset is seen in table 3.2. The percentage in this table is calculated by dividing the number of pixels in all images which are assigned to that specific class divided by the total number of pixels.

Class	**Percentage**	**Weight**
Recording car	5.281%	1.0
Pedestrian	0.052%	102.3
Bicycles	0.089%	59.5
Motorized vehicles	3.212%	1.6
Other	91.367%	0.06

Table 3.2: The weights which are used to train the segmentation network. To prevent the network from prioritizing on the class which is most present in the dataset, losses on classes that are less present get multiplied by a weight. This makes the network respond stronger to wrongly segmented objects from the classes that are less present in the dataset. The weights are determined in a linear way. The weight for the most present moving object class, the recording car, is set to 1. All other weights are scaled in a linear way by computing the difference in the presence of the class. For instance for pedestrians, the weight is $5.281 \times (1.0/0.052)$.

As seen in the table, the most present class of interest in the training dataset is the recording car, as it is al-

ways on the foreground and seen in almost every image tile. The motorized vehicles are a bit less present but pedestrians and cyclists are significantly less present, especially when comparing its presence to the 'other' class which makes up 91.4% of the dataset. Without weighting the loss function depending on the class, segmentation for pedestrians and bicycles is hardly learned as it doesn't have a significant influence on the loss. To weight the loss we set the weight of the most present class of interest, the recording car, to 1. From there, the weights of all other classes are calculated by taking the ratio between the percentages of the presence of the recording car and the other classes. This way, the network can be trained without a bias towards the more present classes.



(a)



(b)



(c)



(d)

Figure 3.3: Segmentation results on street-view imagery. These images are found by tiling the segmentation network over the cubic image as explained in section 4.1.1. Detected objects are marked with an orange overlay.

These were the details of the used dataset and the chosen segmentation network. The training process, training evaluation and the experiments done to implement this network in to the pipeline are described in

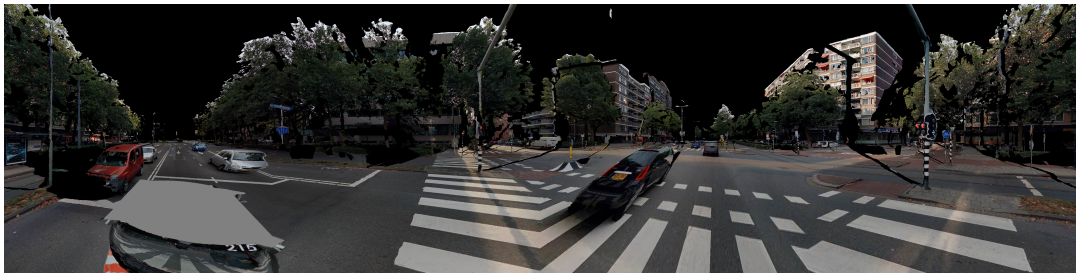
section 4.1.1. Examples of some results of segmenting the four side-tiles of cubic street-view images are seen in figure 3.3. These results give us all objects of interest, which are potentially moving. In the 'combine' block, seen in figure 3.1, the detected objects are separated and one by one they are compared with the pixel-level moving object detection results.

3.1.2. Re-projection

Image segmentation results are combined with pixel-level moving object detection results in the pipeline. Comparing images taken from different angles and finding the differences is often done by visualizing one image as if it was taken from the camera position of the other image. As seen in the previous chapter, figure 2.9, each pixel in one Cyclorama (from now referred to as image **A**) can be matched with a pixel in another Cyclorama which is taken at a relatively close location (image **B**). By repeating this calculation for every pixel, we can transform image **A** and visualize it as if it was taken from the recording position of image **B**. This can be seen in figure 3.4b. This image still holds many undefined pixels, in this thesis shown as black pixels. Black pixels can have two possible causes. The first cause is pixels of which no LIDAR data is available. This mainly applies to the sky and the far horizon. The second cause is occlusions. Some pixels in image **A** describe an object which is not visible in image **B** because there is another object between the object in image **A** and the recording location of image **B**. These occlusions can be found because they result in a miss-match in the depth information, as explained in section 3.1.2. As for these pixels either no depth information is available or the pixels cannot be matched, no conclusions about moving objects can be drawn for those pixels. To fairly compare image **B** with the re-projection of image **A**, all black pixels in figure 3.4b are replaced by the original pixels of image **B**. This results in an image in which the only differences are objects which moved between image **A** and **B**, as seen in figure 3.4c. This re-projection of image **A** is compared with image **B**, which is seen in figure 3.4d.



(a)



(b)



(c)



(d)

Figure 3.4: Visualization of the re-projection process. In this figure, (a) is re-projected to the camera position of (d). First, with geometric calculations, which will be explained later in section 4.1.2, the corresponding pixels between images (a) and (d) are calculated. All images with no depth information or a miss-match in depth (occlusion) are ignored, resulting in the black pixels. To be able to compare the re-projection of (a) with (d) and get only the actual moving objects as results, all black pixels in (b) are replaced by the pixels in (d) at the same locations. This results in (c). For the pixel-level moving object detection, image (c) is compared with image (d). Note that all static objects in (c) are aligned with (d) but the moving objects are in the spots in which they appear in (a)

The most complicated part of this process is finding the corresponding pixels between the two images. For vision-only approaches, this is done by finding corresponding points between the two images and matching those. On the CycloMedia data we don't need to go through the process of finding matching points and rectifying these images because the images are already aligned in a very precise manner. The images are stored in a way in which the horizon is always in the middle of the image in the y direction. As the images are in the cubic projection, the middle of the bottom and upwards facing tiles have to be at $-90°$ and $90°$ respectively. This means the bottom of the four other tiles is at $-45°$ and the top at $45°$. A comparable calculation can be done for the x direction. The middle of the front tile in x direction always points to the north. This also means that the middle of the right, backwards and left facing tiles are respectively at 90°, 180° and 270°. The angles of the edges between the four sideways-facing tiles are then exactly in-between the angles of the middle of the adjacent tiles. An overview of this can be seen in figure 3.5.

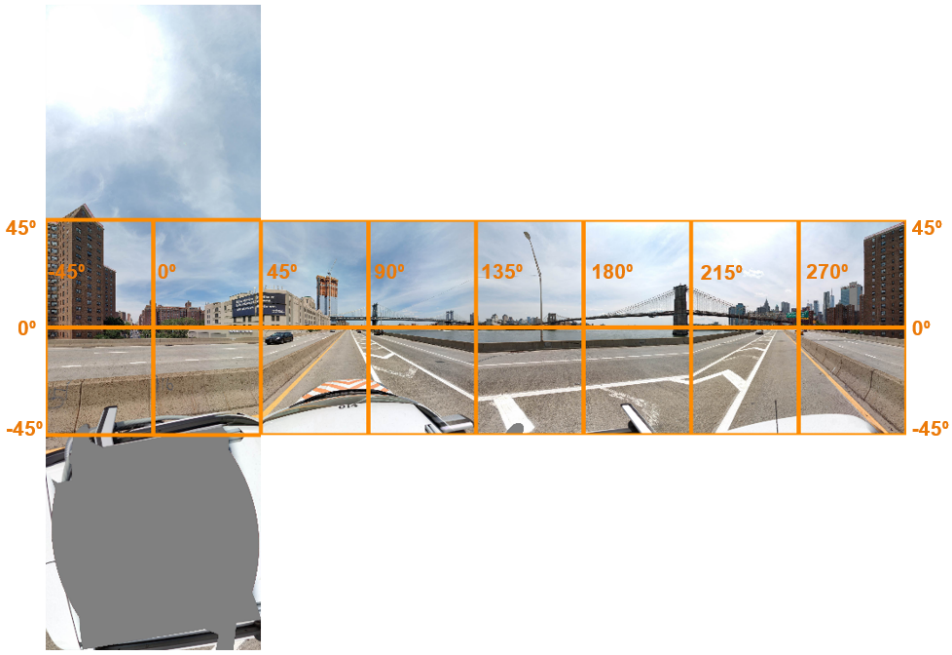


Figure 3.5: A visualization of the alignment of CycloMedia image data. Images which are stored in the cubic projection are always aligned to have the horizon in the middle in y direction and the north in the middle of the front tile (left-most tile) in x direction. Using this information and the knowledge that a cubic tiling scheme is used, all angles of the edges of the tiles can be calculated.

With this knowledge and the available depth information, the re-projection becomes a process of geo-

metric calculations in which pixels can be matched very accurately. To perform the re-projection we need four types of data. First we need two images of which we want to re-project one image to the point of view of the other image. We also need per-pixel depth information for both images. Finally, the recording location and the yaw error of both images should be known. This information is known for all CycloMedia recordings which are recorded with a recording system that includes LIDAR. The recording location, yaw error and a lot of other information of all recordings can be extracted from a big database. The creation of 'depth images', which consist of the depth information of every pixel in the images, is automatically done in the CycloMedia image processing pipeline, which makes them also available for every recording with LIDAR. With all this information all pixels which describe an object which is within the range of the LIDAR can be matched. This matching process consists of the following four steps for every pixel:

- **Pixel to vector calculation.** For every pixel in the first image, a unit vector is calculated which describes the direction from the recording location of the first image to the object which is described in the pixel.
- **Calculation of the real world location.** The exact real-world location of the object in the pixel has to be calculated, using the vector, depth information and recording location.
- **Vector to pixel calculation.** With the real world location of the object and the recording location of the second image, a vector pointing from the second recording location to the real world location can be calculated. With this vector, the pixel location of the object in the second image can be calculated.
- **Occlusion check.** Once a pixel is matched, the real distance between the object and the recording location of the second image can be compared with the depth value for that pixel in the depth map of the second image. If those two depth values don't correspond to each-other, the object seen in the first image is most probably occluded by a different object in the second image.

The pixel to vector calculation is just a geometry calculation based on the alignment seen in figure 3.5. For these calculations, we assume a real world coordinate system in which the positive y axis is pointing to the North, the positive x axis is pointing to the East and the positive z axis is pointing upwards. With the x_p and y_p pixel coordinates and the length of the edges of the cube face L_{face} a three dimensional vector can be calculated and normalized for every pixel as in equation 3.1. Here, x_p and y_p describe the position of a pixel with respect to the top left corner of a cube face.

$$\mathbf{V} = \begin{bmatrix} x \\ y \\ z \end{bmatrix} = \begin{bmatrix} x_p - 0.5 \times L_{face} \\ 0.5 \times L_{face} \\ -(y_p - 0.5 \times L_{face}) \end{bmatrix}, \quad \hat{\mathbf{V}} = \frac{\mathbf{V}}{|\mathbf{V}|} \tag{3.1}$$

Note that, as mentioned before, there is a small yaw error (α). A yaw error is a rotation around the z axis which means the north of the image is not perfectly aligned in the middle of the front face but slightly rotated. Even though this yaw error is very small, the effect of implementing it is slightly noticeable in the results. Therefore, all vectors are multiplied with the rotation matrix which only takes the yaw rotation into account because both the roll and the pitch error are zero:

$$\hat{\mathbf{V}}_{corrected} = R_z(\alpha)\hat{\mathbf{V}} = \begin{bmatrix} \cos\alpha & -\sin\alpha & 0 \\ \sin\alpha & \cos\alpha & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \end{bmatrix} \tag{3.2}$$

Note that for exactly the middle of a cube face where both x_p and y_p are equal to $0.5 \times L_{face}$, the outcome is $\begin{bmatrix} 0 & 1 & 0 \end{bmatrix}^\intercal$. With our above-mentioned coordinate system, a vector pointing in the positive y direction means it is facing North. This means that the $\begin{bmatrix} x & y & z \end{bmatrix}^\intercal$ vector returned from above equations have to be altered depending on which side of the cube is considered. The final $\begin{bmatrix} x & y & z \end{bmatrix}^\intercal$ vector is a combination of the x, y and z values computed in the equations above. The way in which the x, y and z values are shuffled to match the corresponding face size is seen in table 3.3.

Front	Right	Backwards	Left	Up	Down
$\begin{bmatrix} x & y & z \end{bmatrix}^\intercal$	$\begin{bmatrix} y & -x & z \end{bmatrix}^\intercal$	$\begin{bmatrix} -x & -y & z \end{bmatrix}^\intercal$	$\begin{bmatrix} -y & x & z \end{bmatrix}^\intercal$	$\begin{bmatrix} x & -z & y \end{bmatrix}^\intercal$	$\begin{bmatrix} x & z & -y \end{bmatrix}^\intercal$

Table 3.3: The way in which vectors resulting from equations 3.1 have to be rotated to match the agreed real-world coordinate system.

After calculating the unit vector which points from the pixel towards the real object, we want to find the real-world position of the object which is the second step of the process. Real world locations are expressed in a different coordinate system depending on the country. For images taken in the Netherlands, the RD coordinate system (EPSG:28992) is used. Note that the used coordinate system does not matter for the final results, as only the relative distance between the recording locations and the object of interest matters. To calculate the real world position of the object, we use the recording position of the first image ($\mathbf{P}_{r,1}$), the vector pointing from the recording position to the object ($\hat{\mathbf{V}}_1$) and the depth value of the same pixel extracted from the depth images (D_1). Using these values, we can calculate the real world position of the object ($\mathbf{P}_o$) as seen in equation 3.3

$$\mathbf{P}_o = \mathbf{P}_{r,1} + \hat{\mathbf{V}}_1 \times D_1 \tag{3.3}$$

$$\mathbf{V}_2 = \mathbf{P}_0 - \mathbf{P}_{r,2}, \quad \hat{\mathbf{V}}_2 = \frac{\mathbf{V}_2}{|\mathbf{V}_2|} \tag{3.4}$$

With the vector describing real world position of the object ($\mathbf{P}_o$) and the known vector describing the real world position at which the second image is recorded ($\mathbf{P}_{r,2}$), we can calculate the unit vector pointing from the recording location of the second image to the real world position of the object, as seen in equation 3.4. A schematic overview of all symbols mentioned in equation 3.3 and 3.4 can be seen in figure 3.6.

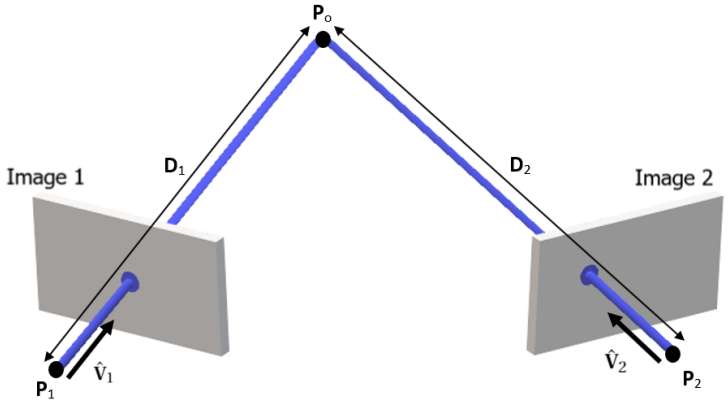


Figure 3.6: A visualization re-projection process. Image 1 and 2 are parts of two panoramic images, describing the same part of a scene. This figure shows how for a pixel in image 1, with corresponding unit vector $\hat{\mathbf{V}}_1$, the real world position of the object shown in the pixel ($\mathbf{P}_o$) can be calculated, using the pixel depth (D_1). From this position we can find a vector pointing from the recording position of image 2 ($\mathbf{P}_{r,2}$) and by normalizing this we get $\hat{\mathbf{V}}_2$. With this vector, the pixel location of the object of interest can be found in the second image.

In the third step of the process, the pixel coordinates x_p and y_p that correspond to the unit vector for the second image ($\hat{\mathbf{V}}_2$) has to be determined. This is similar as the process of going from a pixel to a unit vector, but in a reversed order. First, the unit vector has to be corrected for the yaw error of image 2. This is done using equation 3.2. After that we have to determine in which cube face of image 2 the pixel belongs. This can be done by comparing the x, y and z values of unit vector $\hat{\mathbf{V}}_2$. The biggest absolute value of the unit vector determines the most notable axis. For instance, if the absolute value of y is bigger than the absolute value of x and z, the pixel belongs to one of the cube faces that intersect with the y axis, which are the front and backward faces. If the y value is positive, it means that the pixel is pointing to the positive y axis and therefore the pixel is on the front cube face. Using this logic, one of the six cube faces can be selected based on the unit vector. When the cube face is known the unit vector can be shuffled again, as in table 3.3 and the new unit vector can be transformed back to pixel coordinates as in equation 3.1.

The final step of the re-projection is the occlusion check. When calculating the unit vector for the second image ($\hat{\mathbf{V}}_2$) in equation 3.4, the distance between $\mathbf{P}_0$ and $\mathbf{P}_{r,2}$ is calculated, which is $|\mathbf{V}_2|$. However, it is possible that point $\mathbf{P}_0$ is not visible at all in image 2 because another object is located between point $\mathbf{P}_0$ and recording location $\mathbf{P}_{r,2}$. If this is the case, this object is also present in the depth map of image 2. To check if object $\mathbf{P}_0$ is visible in image 2, we compare the calculated distance $|\mathbf{V}_2|$ with the distance found in the depth map D_2. If D_2 is significantly smaller than $|\mathbf{V}_2|$, the object at location $\mathbf{P}_0$ is not visible in image 2, and therefore we can't check if the object is moving or not. Because the images of interest in this thesis are compared with four other

views in total, the influence of occlusions is limited. The amount of moving objects which are not visible in any of the four views is negligible.

These four steps are performed for every pixel of all four images which are used to find moving objects on the image of interest. This results in a total of four image re-projections. An example of four images re-projected to the point of view of the image of interest is seen in figure 3.7. In this case, the image of interest is image *t*. On this image the moving objects should be detected. To find the moving objects, the image is compared with the re-projection of the two previous and the two next images in the sequence, *t - 2*, *t - 1*, *t + 1* and *t + 2*. This process is explained in section 3.1.3.

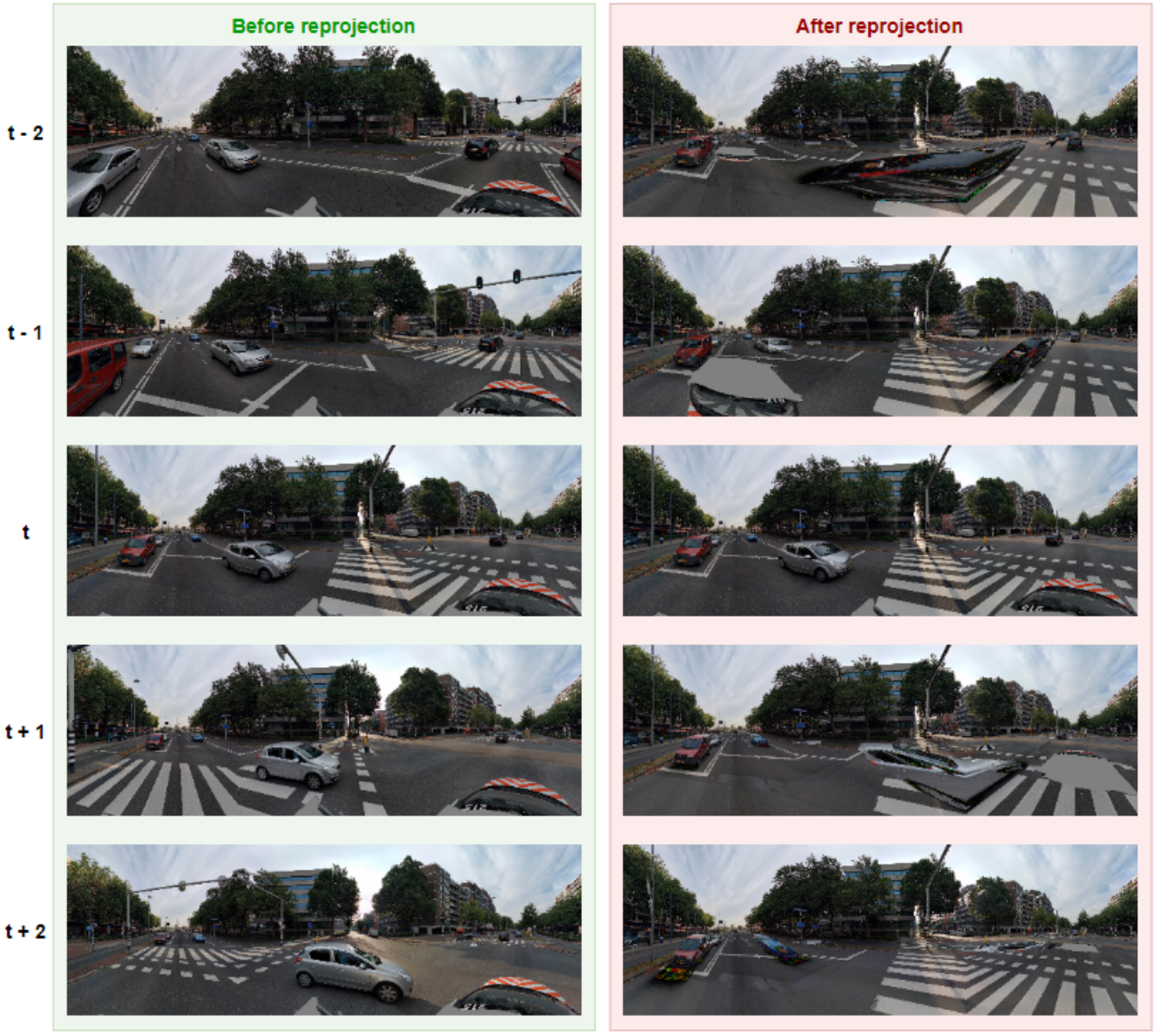


Figure 3.7: The re-projection of four neighbouring view to the point of view of the image of interest. On the left side we see the image of interest *t* and all four images taken within a 10 meter radius. These four images are all re-projected, which means they are visualized as if the images were taken from the recording location of image *t*. As you can see on the right side, this results in images which are nicely aligned with all buildings, road markings and static vehicles at exactly the same spot. The only difference between the images are the moving objects.

3.1.3. Compare re-projections with the image of interest

To compute a pixel-level moving object prediction, the re-projected images (figure 3.4c) have to be compared with the image of interest (figure 3.4d). Several methods are available for comparing image of interest with the four re-projections. The experiments performed on this subject are discussed in section 4.1.3. In the final design, features from SegNet, computed for both images, are used to find the difference between the images. The difference between the image of interest and each of the re-projected images are compared to a threshold, giving a binary score for all four comparisons. Those four scores are summed and then normalized. This results in a final pixel-level moving object detection result in which every pixel has a score in the range $[0,1]$. A score of 0 means that the pixel is not classified as moving in any of the four comparisons and 1 means that the pixel is classified as moving in all four comparisons. The output of this comparison results is represented by the 'Pixel level moving objects' block in figure 3.1. An example of this output is seen in figure 3.9b. In the final part of the moving object detection pipeline, the segmentation results have to be combined with the pixel level moving object detection result. This process is explained in section 3.1.4

3.1.4. Combine segmentation and moving object detection results

Now both the segmentation and pixel-level moving object detection results are available, the two branches of the pipeline can be merged by combining those results. Here the segmentation result is taken as a 'baseline' and from this baseline of objects of interest the static objects are filtered out. Figure 3.8 illustrates this, and based on this figure the process will be further explained in this section. The output of this process is also the final output of the moving object detection part of the pipeline.

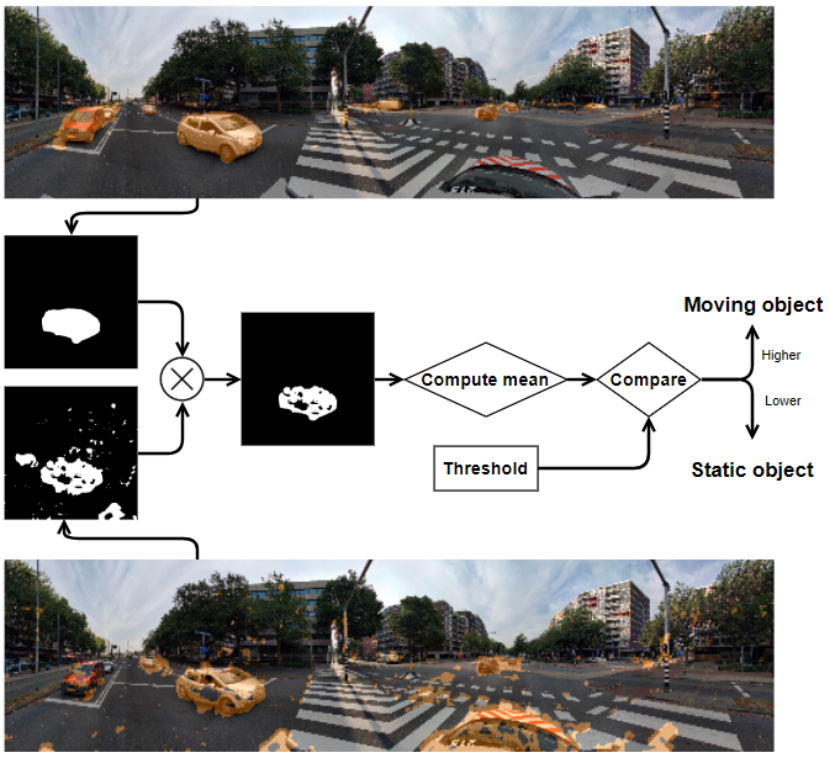


Figure 3.8: The process for filtering the segmentation result visualized. The contours are separated and for every contour the corresponding part of the pixel-level moving object detection is found. The mask of the contour is then multiplied with the pixel-level detection, resulting in only the pixel-level detection for the area of interest. The mean of the values in this area, which range from 0 to 1, is computed and compared to a threshold. If the value is higher than the threshold, the object is classified as moving. If the value is lower than the threshold, the object is considered static and the contour is removed.

At the bottom and the top of figure 3.8 the two inputs are shown. The result at the top is the output of the segmentation branch as explained in section 3.1.1. The result at the bottom is the result of the pixel level moving object detection branch as explained in section 3.1.3. First, all objects of the segmentation result are separated. This is done by using OpenCV to find all separate contours in the mask. These contours are filtered

based on their size and the spatial location of the center of gravity. Contours with an area of less than 50 pixels are filtered out as those are either false detections or objects too far away to be relevant. Another way to filter out false detections is to use the y coordinate of the center of gravity. The y coordinate of the center of gravity should be around the horizon or slightly below the horizon. With this information in mind, detections with a center of gravity far above the horizon or all the way at the bottom of the image can be filtered out as they can be assumed to be false detections. After filtering the contours, the list of remaining contours should be compared with the pixel-level moving object detection results.



(a)



(b)



(c)

Figure 3.9: Example of the combine step which is the last step of the moving object detection part of the pipeline. For each contour in the segmentation result (a), the average per-pixel moving object detection result (b) of all pixels within the contour is computed. This average is compared to a threshold. By doing this, the static objects can be removed from the detections, leaving only the moving objects as seen in (c).

For each contour, first a mask is created in which only the contour of interest can be seen, without the other contours in the image. This mask is multiplied with the pixel-level moving object detection results. This results in a matrix with zeros everywhere around the contour and values between zero and one within the shape of the contour. The sum of this matrix is calculated and then divided by the area of the contour, to make sure the result does not depend on the size of the moving object. This results in a value between zero and one in which zero means that the object is static with a high certainty and one means the object is moving. To get the final classification result the value is compared to a threshold. If the value is above the threshold the object is classified as moving. This means that the contour is drawn on the final result. If the value is below the threshold the object is classified as static and the contour is removed. This threshold is experimentally determined using a moving object ground-truth dataset. This process is explained in section

4.1.4. In figure 3.9, the final result of the combining process is seen. The static objects which are segmented in figure 3.9a, like the parked cars to the left and in the middle, are removed because the segmented objects do not have enough overlap with the pixel level moving object detection. Figure 3.9c is the final output of the moving object detection part of the pipeline. Based on this result, the moving objects are deleted and inpainted as described in section 3.2.

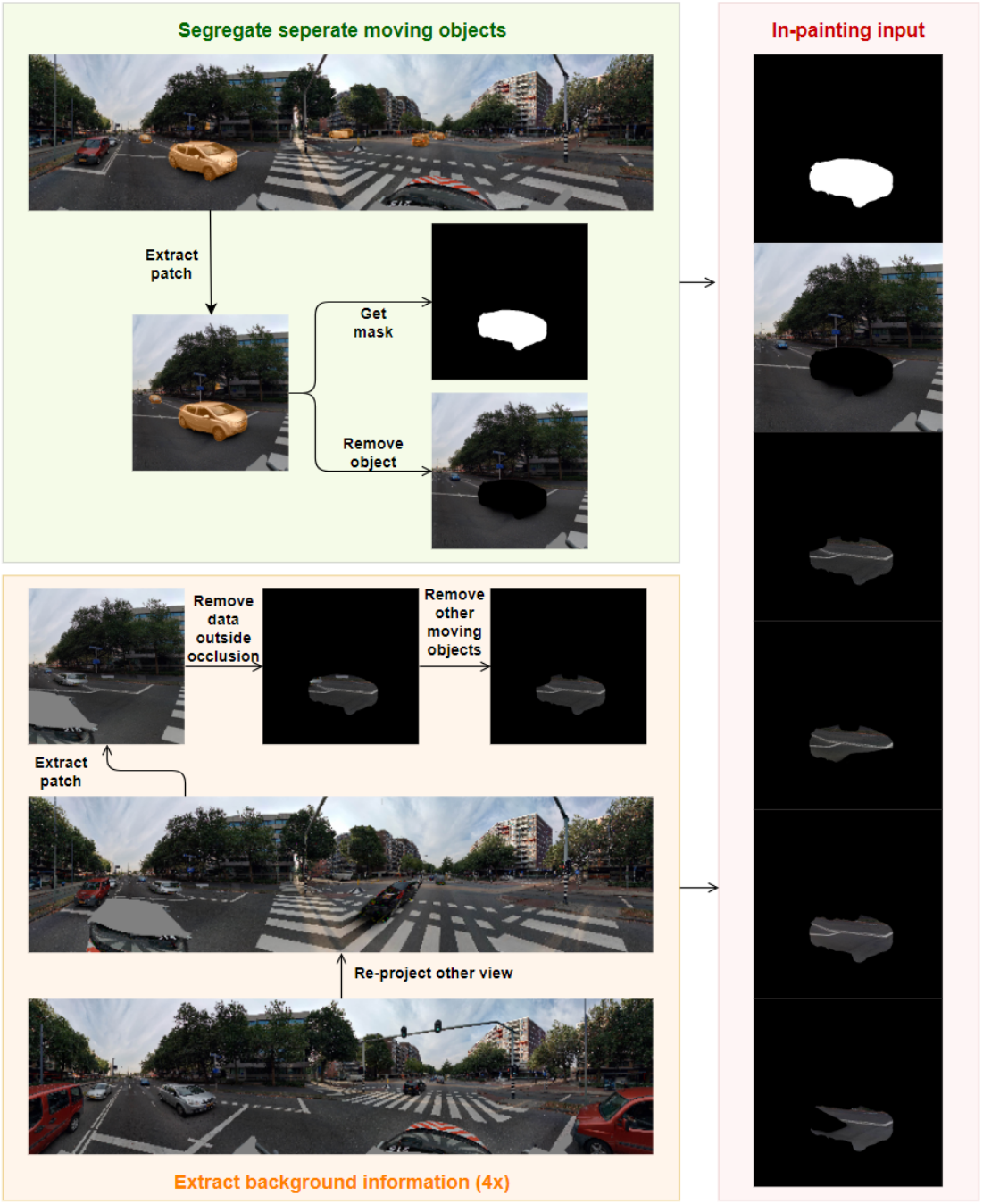


Figure 3.10: An overview of the part of the pipeline that gathers the input data for the image inpainting network. In the segregation part (top left), first one of the moving objects is selected. A patch centered around the object is extracted. From this patch, the moving object is removed and the result of this is an input for the inpainting network, together with the mask of the moving object. In the background information part (top left), four images taken close to the image of interest are re-projected to the point of view of the image of interest. A patch of the same area is extracted. From this patch, first the area outside of the occlusion is removed. After that, the moving objects within the occluded area are removed. After repeating this process four times, the four pieces of background information are fed to the network.

3.2. Image inpainting

After the moving objects are removed the image has to be inpainted. This inpainting should look natural, without any artifacts, and should show the actual content which is behind the moving object. In section 2.3.4 a new approach was suggested using Generative Adversarial Networks. Recent Generative Adversarial Networks (GANs) show a good performance in image inpainting with natural results and few artifacts. Those networks, however, do not use information from other views which means it does not satisfy the requirement of inpainting the actual content. Therefore, for this thesis, Generative Adversarial Networks are combined with information from different views. In this section, the way in which this data is combined is described.

3.2.1. Segregate objects

For the final output of the moving object detection pipeline, which looks like the image in figure 3.9c, the moving objects have to be separated. To do this a binary mask of the detections is used. On this mask, the contours are found by using Open CV to cluster neighbouring pixels. Those clusters are filtered based on the surface area and the spatial locations, to ignore the contours which are not interesting for this thesis. One filter removes the contours of which the area is below a threshold which makes them to small to be a relevant moving object. Next to that, contours which are at the top or the bottom of the image, the areas where no moving objects should be present, are removed as well. This saves time because it reduces the number of iterations of the inpainting network. Now, for every remaining contour we run the inpainting network once, resulting in an inpainting result for every separate moving object. To do this, a 512×512 patch around the object is selected. This patch is centered around the center of gravity of the moving object. The selection of the patch and removal of the moving object is visualized in the top part of figure 3.10. Before feeding it to the network, the patch is combined with information from other viewpoints. The selection of this information from the different point of views is explained in the next section.

3.2.2. Extracting background information

To make an inpainting of the actual background, the network needs information from different views. This information has to be carefully selected. To make sure the information is captured from data which is already warped and rotated into the right perspective, the information of other views is extracted from the re-projected images. These re-projected images are less detailed compared to the original images. This is visible in the blurriness and small artifacts, often caused by LIDAR inaccuracies. For this reason, it is best to limit the amount of information from other views fed to the network. This process is visible at the bottom om figure 3.10 and the experiments which led to this final input type are discussed more in depth in section 4.2.1. First all the information outside of the occluded area is removed as it is redundant and blurry information. Besides that, moving objects often still appear in the re-projected images, mainly in the ones taking 5 meter before or after the image of interest. Those moving objects are also removed from the input data, to make sure the network only receives actual background information.

3.2.3. Generative Adversarial Network

The process of creating the inpainting network consists of several steps. Once the input data type and the desired output is known, the network architecture has to be designed. The preferred approach is to start with a successful existing network and adjust that network to fit the goal of this thesis. To train the network, a dataset is required. As no public dataset is available for inpainting images with a neural network based on information from different viewpoints, a new dataset had to be made for this thesis. When this is done, the network can be trained on the created dataset and tested on the separate test set until the results stop improving. At this point, the weights of the network should be frozen and stored. This stored model then should be implemented into the pipeline. Finally, the pipeline can be used to inpaint images and the results can be evaluated. In this section the input and design of the network is discussed. The creation of the synthetic dataset and the training procedure is discussed in section 4.2.2.

For the design of the network, an existing network architecture is used as a baseline. The first big publication on using GANs for image inpainting was Context Encoders, published by Pathak et al. [33]. After that, three main improvements were made on this network by other publications. Iizuka et al. [34] replaced the

fully connected layer with dilated convolutional layers which made the network more efficient. They also used two discriminators in their work. The local discriminator only looks at the inpainted area to verify if the inpainting is correct. The global discriminator looks at the whole image and verifies if the inpainted part fits into the whole image without leaving artifacts. These two discriminators were also used by Zhao et al [35] and Song et al. [36]. Another improvement made on Context Encoders was the use of a second network. This approach was used by Zhao et al [35] and Song et al. [36]. This second network is usually referred to as the refinement network and takes the output of the first inpainting network and improves the result. All these improvements where used by Yu et al. [37]. Their network reached state-of-the-art scores by combining all the three above mentioned separate improvements into one network. This network does not only show state-of-the-art performance, it also shows results on images with bigger resolution compared to the other images. As they also published their implementation, coded in TensorFlow, this is the ideal network to use as a baseline. However, some modifications were made to utilize this network for the goal of this thesis. Three main changes were made:

- **Changing the spatial dimension of the input, output and occlusions.** The input of the network of Yu et al.[37] is a 256×256 mask and an image with the same dimensions with a hole in it which should be inpainted. The output is an inpainted image, also with the same spatial dimensions. The holes which have to be inpainted have a maximum size of 128×128. As images at CycloMedia have a high resolution, it is beneficial to be able to inpaint images of a higher resolution. Therefore, the spatial dimensions of the input and output of the network was increased to 512×512 by adding two more hidden layers at the start and at the end of the network. To be able to learn the inpainting of larger moving objects, the size of the inpainted area is increased to 256×256. This makes the inpainting task significantly harder, but giving information from other view-points makes the inpainting task easier which makes up for it.

- **Changing the number of channels of the input.** As described above, the input of the network is a 3-channel RGB image and a binary mask with 1 channel. This makes the input of the network of Yu et al.[37] $256 \times 256 \times 4$. Next to scaling up the spatial dimensions of the input, we also want to give more information, from four different views. This extra information is concatenated to the other inputs which means another $4 \times 3 = 12$ channels are needed. This results in the final input of the new network with dimensions of $512 \times 512 \times 16$.

- **Removing the secondary network.** In most publications which use a secondary network, the secondary network was described as a refinement network. After an initial inpainting was made, the secondary network would find patches in the other areas of the image that look like a patch of the initial, blurry, inpainting and use those patches to give the inpainting more detail. In the case of this thesis, however, the information should be taken mainly from the other views and not from the surroundings of the occlusion. As this information from other views already gives a lot of information about the background, a secondary network is less necessary. Besides that, the second network makes the network slower, which is undesirably as the inpainting network can be used multiple times per image. From experiments done for this thesis, it turned out that the result of the first network becomes just as good as the output of the second network after training for a long enough time. Therefore, a decision was made to remove the secondary network, which helps the optimizer to focus more on the training of the first network and therefore the first network will also perform better.

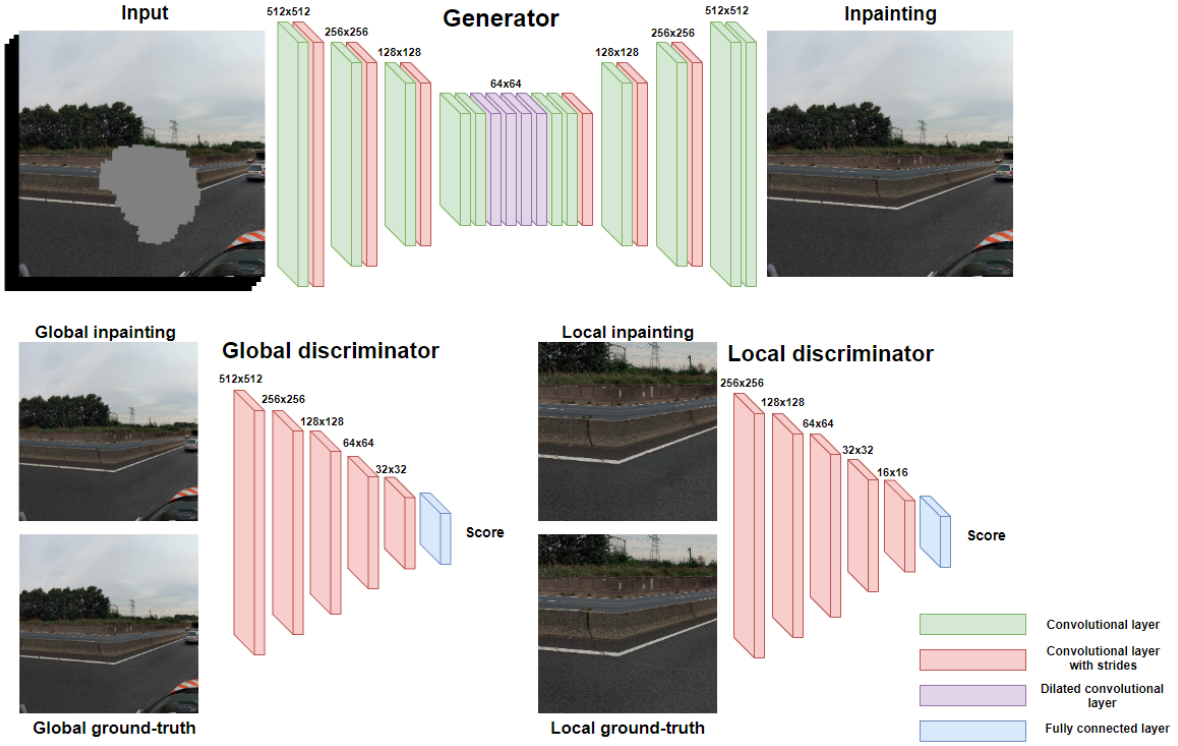


Figure 3.11: Schematic overview of the image inpainting network. The generator takes the input, consisting of the image of interest, the mask and the information from other views and makes an inpainting of the hole in the image of interest. The global discriminator compares the full output with the ground truth. The local discriminator compares only the inpainted area with the ground truth. These discriminators are only used during the training procedure. Note the quality of the inpainting. The only noticeable difference is the blurriness of the barrier.

The final resulting network turned out to be almost completely different from the network of Yu et al. [37]. The main aspects which remain from the original network are the dilated convolutions, most of the training parameters, and the local and global discriminator which only have one extra layer because of the increased spatial dimensions. The schematic overview of the network can be seen in figure 3.11. As seen in this figure, the input consisting of 16 channels is fed to the generator network which has an encoder-decoder structure. In the middle of the encoder-decoder structure dilated convolutions are used to interchange information between different areas of the image. The output of the generator is fed to the global and local discriminator. These two discriminators compare the output with a ground truth sample and gives the generator network feedback during training. To perform this training, a dataset with available ground-truth is necessary. The way in which this synthetic data set is generated is described in section 4.2.2. This section also discusses the training process and the experiments which are done with different types of input data. The final trained model is implemented in the pipeline, which finalizes the pipeline. In figure 3.12, two examples of the full pipeline is shown, with the input images in figure 3.12a and 3.12c and the output images in figure 3.12b and 3.12d. In appendix B and appendix C many more results of the full pipeline can be seen, showing results on varying objects in different scenes and results on sequences of cycloramas respectively.



(a)



(b)



(c)



(d)

Figure 3.12: Image (a) and (c) are examples of images of interest which are fed into the pipeline, together with the other information. Image (b) and (d) are the output corresponding to input (a) and (c). The same image, but without moving objects.

4

Experiments

In this section, the main experiments will be presented. The presented experiments and methods led to the creation of the final pipeline design. For each part of the pipeline some visual results are shown. For most parts it is also possible to get a subjective measure of the results. In this case, the used metric is shortly explained and the values are presented. In this chapter, the experiments and results are described in a similar way to the previous chapter, divided in small parts comparable to the blocks in figure 3.1. These parts are divided in two sections, starting with the moving object detection part in section 4.1, followed by the image inpainting part in section 4.2

4.1. Moving object detection

The moving object detection part is the most extensive part of this pipeline. For the experiments, several different types of neural networks were used, as well as more conventional computer vision approaches. First, the image segmentation part is discussed in subsection 4.1.1. Secondly, in subsection 4.1.2, the performance of image re-projection is discussed. Thirdly, the experiments and results on comparing the other points of view with the original image are discussed in subsection 4.1.3. Finally, in subsection 4.1.4 the details on combining the segmentation with the pixel-level moving object detection results are discussed.

4.1.1. Image segmentation

In the first part of the pipeline we aim to find all objects of interest, which are objects of classes which are possibly moving. From the objects that are found in this part, the moving objects will be selected later on in the pipeline with help from the pixel-level moving object detection branch. Several steps are required to get nice segmentation results on the CycloMedia data. In section 3.1.1, the choice and adaptation of the segmentation network and the adaptations of the dataset have been discussed. Now, the network should be trained and regularly the weights should be tested to find the best training result. The weights of the best training iteration should be kept and can be placed into the pipeline. In this section the training results and experiments on implementing the network into the pipeline are shown.

As discussed in section 3.1.1, we choose a network, simplified the dataset and weighted the dataset to put more focus on the classes which are less present in the dataset. Now the network and the final dataset is ready, the network can be trained. Before starting the training session, it is important to split the dataset into a training and an evaluation set. When training a network on a dataset, it does not only learn the general concept of image segmentation on the given data type, the network starts to overfit on the given dataset when training is continued for too long. This means that it learns to segment the training images very precisely, but the segmentation performance on images which are not in the training set degrades. This is the reason for separating a part of the ground-truth as an evaluation dataset. For this thesis, the data is split into 70% for training and 30% for evaluation. This results in 2800 training samples and 1200 evaluation samples. During the training of the network on the training set, the weights of the network are stored every 5.000 iterations. Those weights are used to segment all the images in the evaluation dataset. As for those images ground-truth

data is also available, the loss of the segmented evaluation images can also be calculated. Based on this loss, we can calculate how well the network is performing on images which it actually hasn't seen before. Note that while running the evaluation, the weights are not updated and therefore the network can not overfit on the evaluation set. By training like this, two loss curves can be made. The curve for the actual training, which uses the cross-entropy loss, will keep converging to zero as the network starts to overfit and learns the perfect segmentation for the dataset eventually. The evaluation curve converges for a while but the evaluation loss will start going down at some point. This results in a maximum value in the evaluation curve which denotes the peak performance and the point where the network starts to overfit on the training set. The weights from this maximum evaluation loss are kept as finalized weights.

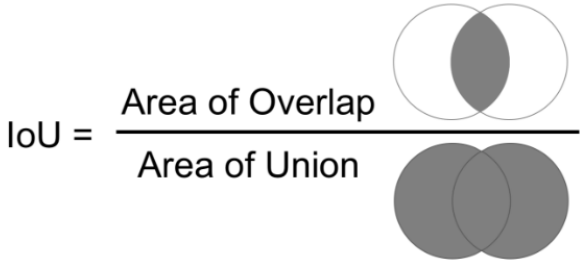


Figure 4.1: The mean Intersection over Union (mean IoU) algorithm used to measure the quality of the segmentation network. The mean IoU measures the correspondence between ground-truth masks and predicted masks. The area of overlap is divided by the combined area of the ground truth and predicted mask. The division converges to 1, which means the predicted mask and the ground-truth mask are perfectly aligned.

Class	**Mean IoU**
Recording car	0.960
Pedestrian	0.157
Bicycles	0.145
Motorized vehicles	0.666
Other	0.986

Table 4.1: The mean IoU of the final checkpoint over the different classes. As seen in the table, there is a major difference between the different classes. The difference is significant and seems to be roughly proportional to the presence of the classes in the dataset as shown in table 3.2

For testing the final segmentation results, the mean Intersection over Union (mean IoU) loss is calculated. The mean IoU is a metric to measure the correspondence between a ground-truth bounding box (or mask in this case) and a predicted bounding box. It is calculated by calculating the overlap (intersection) of the predicted and ground-truth masks and dividing it by the union, which is the total surface of the two masks combined. Translating this to computer vision related terminology, this means the true positives are divided by the sum of the true positives, false positives and false negatives. The formula is illustrated in figure 4.1. This metric returns values in the range $[0, 1]$ where 1 means that the predicted and ground-truth mask are perfectly aligned. For every stored checkpoint, the mean IoU is calculated for every image in the evaluation ground-truth dataset. The checkpoint for which the best mean IoU value is returned is stored. Because of the way this segmentation network is trained, it is not possible to get a full graph of the training progress. As the segmentation dataset was made concurrent with this thesis, a small part became available every time. The network was trained on those initial parts of the final dataset and once a bigger sample became available the training was continued on the bigger datasets. The final result is the checkpoint which gives the highest mean IoU for the final training phase, on the full dataset. The best final result was achieved after training for a total of 460k iterations. This training process was paused several times and continued on a new dataset with an increased size. Therefore, no representative graph of the training and evaluation process can be shown. In total, training took about two weeks on a Tesla P100 12 GB GPU. The mean IoU on the evaluation dataset after training for 460k iterations is 0.583. The distribution of this mean IoU over the separate classes can be seen in table 4.1. There is a major difference between the scores of the separate classes. The two main influences on the separate classes seems to be the class distribution in the dataset and the complexity of the data. The recording car is the most present class of interest in the dataset and also has a high score. Motorized vehicles

are slightly less present in the dataset but have a higher complexity and therefore a significantly lower mean IoU. Clearly, not enough pedestrians and bicycles are available in the dataset to get a good segmentation of this class. In figure 4.2, some examples of the segmentation ground truth (left) and final segmentation result (right) on samples from the evaluation dataset are given. Note that these images were never seen by the network during training.

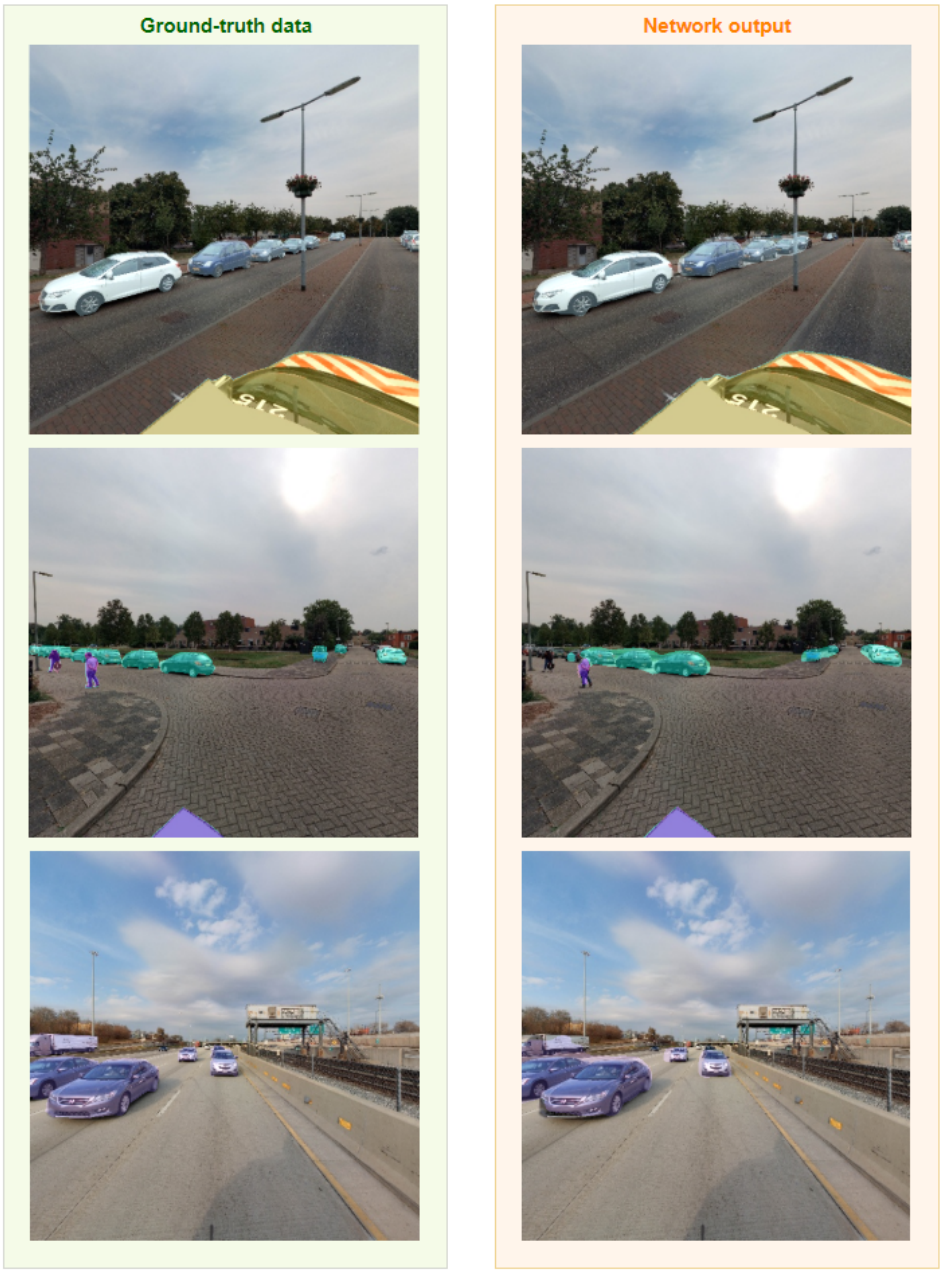


Figure 4.2: Results of running the trained segmentation network, after 460k iterations, on images from the evaluation dataset. On the left side is the ground-truth data and on the right side the output of the segmentation network. As seen in the images, the network is able to find most cars but is not completely accurate at finding the outline of the car. The recording vehicle is also segmented very well. For pedestrians and cyclists, the network less accurate. This was to be expected, because of the relatively small dataset and the low number of pedestrians that appear in the data.

Now the network is trained and the final weights are available, it can be implemented in the pipeline. In the pipeline, segmentation masks should be generated on an image consisting of four faces (front, right, back, left). These four faces have a resolution of $1536 \times 1536 \times 3$ in the pipeline, but the segmentation network is

trained on $512 \times 512 \times 3$ images. The easy solution to this is to simply scale down the four faces by a factor of 3 which results in exactly the right dimensions for the segmentation network. A problem, however, is that the segmentation network is trained on a dataset in which only objects within a radius of 40 meters from the recording vehicle are annotated. This means the network learned to segment the objects within that range only. Even though the objects within this range are the most relevant objects, because they occlude a bigger part of the scene, it is still beneficial to also find the objects further away. A good solution for this is to zoom in on the area around the horizon and feed these images to the network as well. In those tiles, the objects that are further away appear bigger and therefore can be recognized by the network.

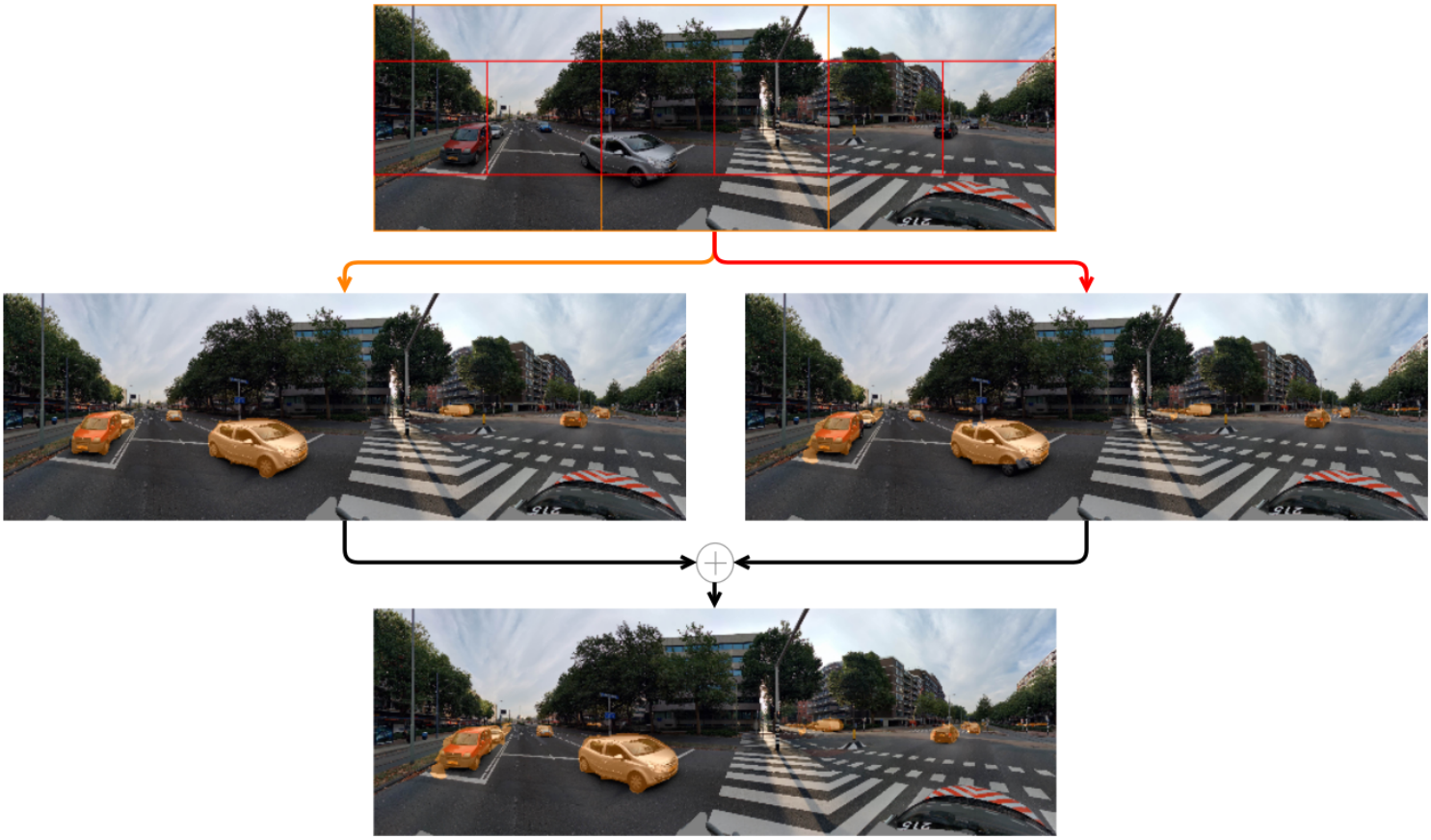


Figure 4.3: The combination process of the results at two different scales. As can be seen in the image on the left side, most of the relevant objects are already found by segmenting the large scale (orange) tiles. By segmenting the small scale (red) tiles, a few more objects are usually found around the horizon. These objects are added to the objects that were already found on the large scale, resulting in the final segmentation result, shown at the bottom. In this image, the extra information gained by segmenting on the smaller scale are the line of parked cars slightly to the right of the middle of the image and some parked cars behind some vegetation in the far right of the image.

Another experiment performed during the implementation of the segmentation network was the amount of overlap between tiles. As some moving objects are centered around the border between two tiles, both tiles only show a part of the moving object. For the network, these parts might be harder to detect compared to whole objects. A solution to this problem is using overlap. For this experiment, an overlap of 20% between tiles is used. Within the overlap area, each pixel that is classified as an object of interest by at least one of the two segmentation outputs is classified as an object of interest. This tiling scheme, however, did not show any improvements in the results. This is most probably caused by the edges between tiles. When using images of the cubic projection, straight lines in the real world are also straight in the images, except between two different tiles. On those edges, the two straight lines from the two tiles are connected, but don't have the same angle of incidence. This means that, when using overlap, the network receives images in which objects can have an unusual shape. As neural networks partly focus on the shape of the objects and the network is only trained on full tiles without overlap, this is not beneficial for the final result. Segmentation results with and without 20% overlap were compared for a large number of images and based on this, a decision was made

to not use overlap in the pipeline. Not only does it increase the computational cost, it also does not show improved results. In the top image in figure 4.3 the final tiling scheme, with 2 scales and no overlap, is shown. This results in a total of 12 image tiles, 4 at the big scale and 8 at the small scale. These 12 tiles can be fed to the network in one batch.

The output from the network, 12 separate segmentation results, has to be combined into a segmentation mask of the full image. Tiling the 4 results at the big scale gives a segmentation mask of the full image. The area around the horizon has to be updated, based on the result of the 8 smaller scale segmentation outputs. For this area, all pixels which are detected as object of interest based on the bigger scale are kept and pixels which are detected on the small scale but not on the big scale are added to the final result. This means the overlap areas between the two scales are combined in a 'max' approach. In figure 4.3, the separate results of the two different scales are shown as well as the result of combining those. The combination of those two scales is the final result of the segmentation part of the pipeline.

4.1.2. Image re-projection

In section 3.1.2, the process of image re-projection on the CycloMedia image data was explained. This section focuses on testing the accuracy of the re-projections. To test this, the re-projections are compared with the original images using three different metrics. This test is performed for different re-projections over different distances to test how well the re-projection quality decreases when the baseline between the images increases. Besides the baseline distance, the quality of the 3D mesh which is used to get the depth information is also an important factor in the performance of the re-projection process. Some examples of issues in the 3D meshes are shown and explained in this section.



(a) (b) (c)

(d) (e) (f)

Figure 4.4: Examples of LIDAR and meshing problems that influence the quality of the re-projection. Image (a) and (b) show how LIDAR goes through windows and measures whats behind the window. Notice the difference between normal windows and windows with blinds behind it. Image (b) and (e) shows the smoothing effect of the meshing process. In the mesh the car is connected directly to the ground. Image (c) and (f) shows some results for poles. Poles usually show up quite different in the meshes compared to the images.

The main cause for artifacts in the re-projects are problems in the LIDAR recordings or the meshing process which is performed on the LIDAR point-clouds. One of the main causes for artifacts in the re-projected images are windows. LIDAR beams go through the windows and therefore the background of the window,

which is often the inside of a building. For parked cars this problem is even more bothersome for this thesis. Parked cars should show up at exactly the same spot between re-projected images and therefore not be detected as moving object, but cars can still appear quite different because of LIDAR artifacts caused by the car windows. Another cause for artifacts are thin structures. Thin structures like traffic sign poles or fences are often badly recorded with LIDAR or which usually makes them show up much thinner in the 3D mesh. Finally, the meshing process itself is also a cause for artifacts. The meshing process has a slightly smoothing effect. This results in loss of details, for instance in small gaps being closed. Examples of those three causes for artifacts can be seen in figure 4.4.

Another influence on the quality of the re-projection is the baseline. Images taken further apart have a decreasing accuracy. To test how far the re-projected images are still accurate enough, a set of 20 images where selected. For all images, the next four images in the sequence ($t+1$, $t+2$, $t+3$ and $t+4$) where re-projected to the point of view of image t. For the validity of this test, scenes without moving objects where chosen, which means that the re-projected images should in theory look exactly the same except for the location on which the ego-vehicle shows up in the image. For those re-projections, the Peak Signal to Noise Ratio (PSNR), L1 loss (mean absolute difference) and L2 loss (mean squared difference) was calculated. The PSNR is a metric to measure the noise in an image. We expect a higher noise in re-projections with a higher baseline which results in a lower PSNR. The percentage of matched pixels is also calculated. As discussed before, areas outside of the range of the LIDAR scanner (sky, horizon) can not be matched as there is no depth information available for those pixels. In practice this means that roughly 30-35% of the pixels cannot be matched. From the pixels that do have depth information, some of the pixels cannot be matched with the other image as the object in the pixel is not visible from the other point of view. The number of pixels that are occluded in other views is expected to increase with a higher baseline, resulting in a lower percentage of matched pixels. Based on the experiment on the set of 20 images, the graph seen in figure 4.5 can be made.

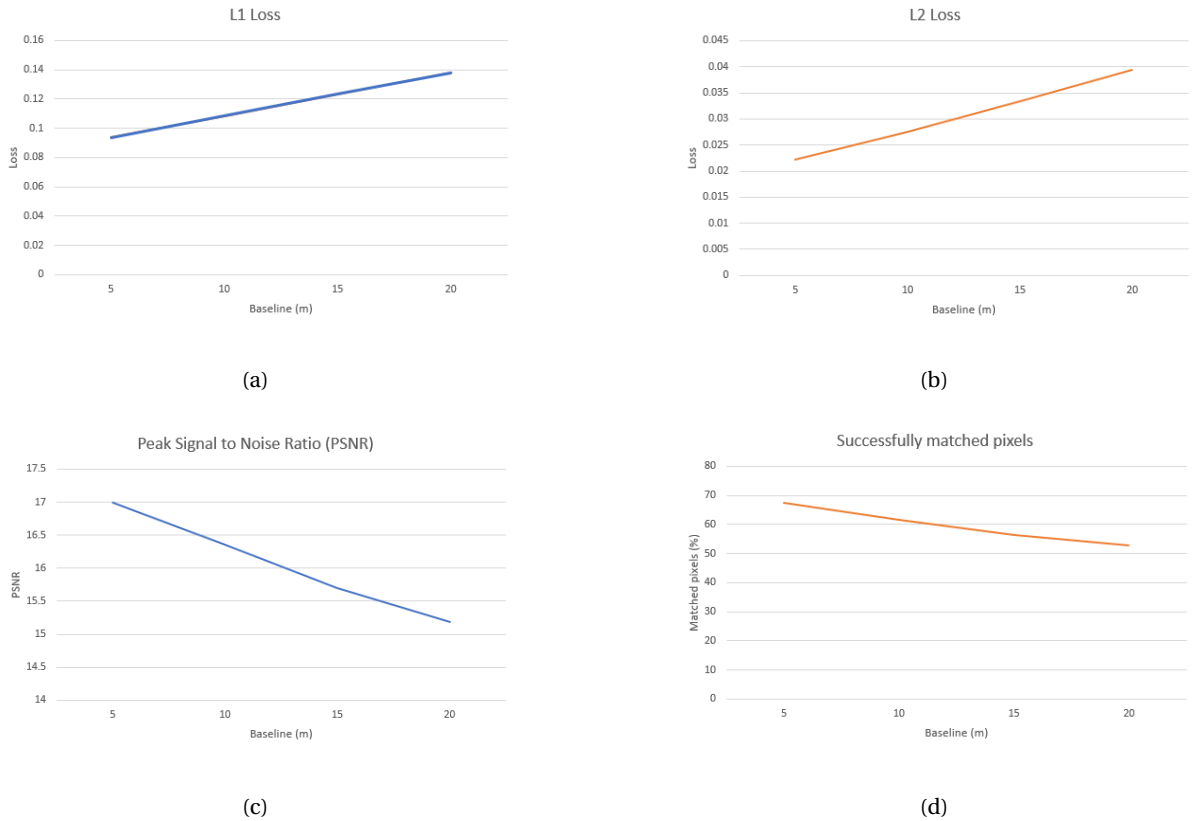


Figure 4.5: Quality comparison of the re-projected images with an increasing baseline. Both the L1 (a) and L2 (b) loss increases with an increasing baseline indicating a decrease in quality and the PSNR (c) decreases which also means that the quality decreased. Furthermore we can see that an increasing baseline significantly reduces the amount of pixels that can be matched. This means that a higher baseline results in more pixels from one image being occluded in the other image.

Both the L1 and L2 loss increase linearly when the baseline is increased. The Peak Signal to Noise Ratio decreases, which means the images become more noisy and therefore also proofs the decrease in quality. Another problem with the increasing baseline is the increasing rate of occluded pixels, which drops significantly when decreasing the baseline. Besides these factors that lead to a decrease in re-projection quality, using more re-projections also leads to a higher computational cost as the re-projection map has to be calculated several times. By only using the two nearest views (5-meter baseline) only the re-projections with the best quality are used but the result is only based on two images. For objects moving at a slow pace, the objects might not have moved enough between two recordings to detect them as moving objects. Therefore, only using two views might not be significant enough. This means the optimal amount of neighbouring views is a trade-off between significance on the one hand and a reduced re-projection quality and a higher computational cost on the other hand. After qualitatively comparing results for several images using a different number of neighbouring views each time, a conclusion was drawn that using four different views is enough for the desired moving object detection result and using more views is mainly an increase in computational cost. The way in which those four re-projections are used for pixel-level moving object detection is discussed in the next section.

4.1.3. Comparing different views

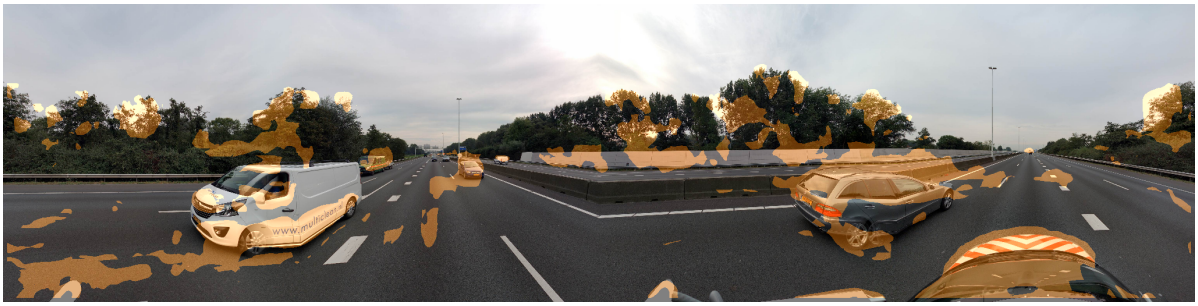
After the re-projection process is finished we have four images recorded at different locations, all seen from the point of view of the image of interest, as seen in figure 3.7. For the next step in the pipeline, these four re-projections should be compared to the image of interest. This comparison should result in a pixel-level score which mainly highlights the moving objects, as those are the main differences between the images. There are many different ways of comparing images with lots of options to choose from. One could for instance use different color spaces (RGB, HSV) or combine them, utilize histograms or consider patches around the pixel of interest with different sizes. Many experiments were performed on the comparison of images, which won't all be treated in this section. Some of the main experiments will be presented instead. First, for the more conventional image comparison methods some experiments on the RGB and HSV color space are shown. After that, a more advanced way of comparing images is discussed which is based on intermediate feature maps extracted from neural networks. This method which became popular only recently is explained and the experiments and results of this method are shown.

The comparison is not performed on the highest resolution of images which is available, due to a strongly decrease in computational cost. The image data on which the comparison is done is still of a high resolution though, which is roughly 14.1 MP. Because of this high resolution, pixels could quite easily be mismatched by a small margin of one or a few pixels. Therefore, directly comparing pixel to pixel does not give good results, as there might be a small shift which means the pixels do not exactly describe the same object. A solution to decrease the effect of these small shifts is comparing patches around every pixel. Comparing patches around pixels instead of just the pixels itself also has an averaging effect, which makes the result less susceptible to outliers. The desired kernel size k of these patches is a trade-off between speed and accuracy. By increasing the kernel size of the patches the computational cost increases but, up till a certain point, the accuracy increases. Results of comparing four re-projection maps with the image of interest, based on RGB and HSV color space, are shown in figure 4.6. After a lot of experiments with different conventional image comparison methods, the conclusion was drawn that those methods do not give the best possible results. There are two main problems noticeable when comparing just based on color values. One problem is large number of false detections on trees and asphalt. Both trees and asphalt are very noisy in images. As the re-projection map is not fully accurate, small patches containing trees or asphalt can be quite different. The second problem occurs when objects have a similar color to their background. This problem is also visible in figure 4.6. The car in the right half of the images has a similar color compared to the asphalt. Because of that, the bottom of the car is not detected in both of the images. Because of those two major problems, experiments were done on other methods on comparing image patches.

In recent years a lot of research has been done on the features of neural networks. Neural networks always have been some sort of black box and people do not know exactly how the input is translated into the output. To be able to understand and train neural networks it is better to be able to visualize what those intermediate results look like. An interesting read on the state-of-the-art on feature visualization was published by Olah et al. [40]. Recent publications have used those intermediate results, the values in hidden layers in neural networks, for varying purposes. Usually the output layer of a neural network is a relatively small vector or



(a)



(b)

Figure 4.6: Results of using the RGB colorspace (a) and the HSV colorspace (b) for comparing the image of interest with the four re-projected images from other viewpoints. The two main problems that occur when comparing just on color values are clearly visible. The trees and asphalt are both very noise and therefore often detected resulting in false positives. Objects with a similar color to their background, like the car on the right, are often not detected resulting in false negatives

matrix with the final score. Intermediate outputs are much more dense as hidden layers usually have a big number of different filters, often up to 256 or 512 filters in a layer. This means that, by extracting those intermediate results, a much more dense description of an image can be gathered, which includes information on for instance shapes, texture and colors. Those feature descriptions of image have been used recently with success for different use-cases. Arandjelovic et al. [42] uses this principle for place recognition. They use a neural network for feature extraction, and use the features of the buildings to compare it with other features in buildings, with help of their own layer which is called VLAD layer in the paper. For this research, they tested several different neural networks, but got the best results when using the VGG network [3]. From this network, they got their best result by extracting the features of the fifth hidden layer. Another implementation of feature comparison is published by Minaee et al. [43]. In this publication, the features are used for iris recognition. Features are extracted by feeding iris images to the VGG network and taking the output of one of the hidden layers. In this work a comparison between different hidden VGG layers is included and the best results where found by using the tenth hidden layer, which is different from the best layer of Arandjelovic et al. [42], which suggests that the output layer with the best features is not always the same.

As both publications get their best results by using VGG as their network to extract features from, this is also the one which was tried first for this thesis. As no CycloMedia dataset was available at that time, a pre-trained model was used to extract features. This means the weights are used that were saved after training the network for a very long time on the dataset for the ILSVRC-2012 ImageNet classification challenge [41]. The VGG network was used to extract features from both the re-projected images and the original image. The way in which features are compared is described further into this section together with all implementation details. After visual inspection of results the best results were achieved by using the 8th hidden layer of VGG-net as the output result. An example of an result can be seen in figure 4.7a.

Even though the results were quite good, there was still lots of room for improvement, both in the quality of the results as well as the computational efficiency. For every image for which a feature description has to

be created, the image has to be divided in small 224×224 tiles, which is the input size of the VGG network. These small tiles have to be fed to the network which is a computationally heavy network, and then the tiles should be put together again to get a feature description of the whole image. When the segmentation dataset became available during this thesis, the pixel-level moving object detection results were combined with SegNet as described in chapter 3. This means that every image is passed through two networks, SegNet to get a segmentation result and VGG to extract features for the pixel-level moving object detection. The pipeline could be simplified and could be made more efficient by using SegNet for both segmentation and feature extraction. This means both results can be received from one iteration of the neural network. A result of using SegNet for feature extraction can be seen in figure 4.7b.



(a)



(b)

Figure 4.7: Result of using features from VGG network (a) and SegNet (b) for comparing the image of interest with the four re-projected images from other viewpoints. As can be seen, the feature comparison show a great improvement over a simple color-based image comparison. In these images, different intensities of the orange overlay are visible, in which a higher intensity means a higher certainty of movement. Both networks find all moving objects in the image, but SegNet finds the moving objects with a higher certainty.

The implementation of SegNet for extracting feature maps was a bit more convenient compared to VGG. As described in section 4.1.1, the tiling for the segmentation network was done on two different scales, which is shown in figure 4.3. Because the images are also tiled for segmentation the tiling for feature extraction is not needed. Next to that, the iteration of the segmentation is also done already, making the extraction of the feature map from the network also free. The only computation for the extraction of the feature maps is the combining of the tiles to form the final full-size feature map. Next to that, the SegNet network is trained on CycloMedia dataset which means it has learned to recognize the vehicles of interest. The VGG network is pre-trained on a public dataset with different types of images and different classes. This means both networks are looking at colors, textures and shapes, but only SegNet recognizes the relevant textures and shapes which belong to classes of interest. All these reasons resulted in a decision to keep using SegNet for the feature extraction.

After the feature maps are extracted, the comparison process can be started. In this process, the feature map of the image of interest is compared with the four feature maps of the re-projected nearby images. Each feature map is of size $h \times w \times c$, in which h and w are the height and width of the image and c is the number of filters. If we assume layer 4 as an output, the dimensions of the height and width are scaled down by a factor of 8 and the number of filters is 512. This results in a feature map of $64 \times 256 \times 512$ for the four tiled cube faces. These feature maps should be compared and a pixel-level result should be obtained, which means the matrix

should be reduced to $64 \times 256 \times 1$. To do this, the mean absolute error (L1 loss) is used. This loss function is chosen because it is cheap and less sensitive to outliers compared to the mean squared error (L2 loss). This means that for all pixels, the two corresponding vectors of 256 values are subtracted, and the mean of the absolute value of this subtraction is the final result. This results in a matrix of a mean absolute error for every spatial location.

These matrices of mean absolute errors of the four different comparisons (the feature map of the original image with the four feature maps of the re-projections) have to be combined to get a final prediction. For this combining process several approaches have been tried. Finally, we choose to separately compare those matrices to a threshold and sum and normalize the result. Each matrix is compared to a threshold, of which the value is experimentally determined and dependent on which SegNet output layer is used, resulting in a matrix of zeros and ones. These four matrices are summed and divided by four, which leads to one $64 \times 256 \times 1$ matrix in which each pixel has a score between 0 and 1 with 0.25 intervals. This matrix is scaled back up by the factor of 8 using bi linear interpolation to get back to the original 512×2048 size of the input. This is the final matrix which is used to combine the segmentation with the pixel-level moving object detection, as explained in section 4.1.4.

As mentioned above, SegNet was chosen for the final feature extraction because of various reasons. Not only is using SegNet much more efficient, SegNet is also trained on CycloMedia data and therefore the network should be better at getting a feature description of the images. The next choice is about which hidden layer in SegNet should be chosen for the best features. To objectively judge the quality of the feature maps from different layers, a ground-truth for moving objects is needed. This is not available because the annotated dataset does not distinguish between moving and static objects. To quantify the results of both the pixel level moving object detection and the final output of the moving object detection pipeline (section 4.1.4), a small dataset of 30 images has been made using images from the annotated segmentation dataset. In this dataset all annotated objects of classes which might be moving (car, pedestrian, cyclist, truck ...) are manually checked to see if they are actually moving or not. Only the annotated objects which are moving are kept resulting in a small dataset of moving object ground-truth.

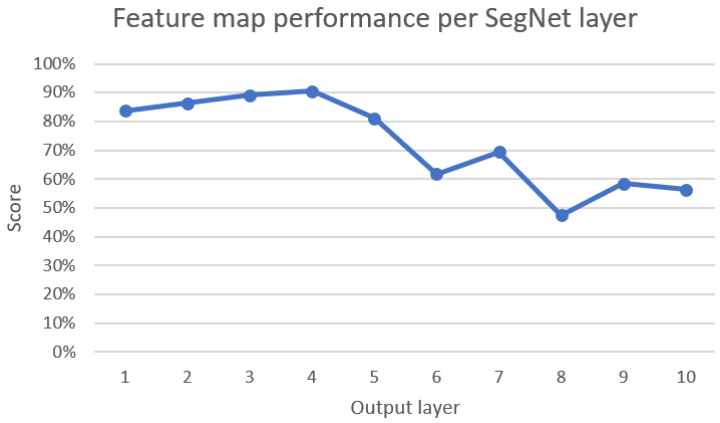


Figure 4.8: For each output of the 10 blocks in the SegNet architecture, the feature maps are tested to see which feature maps performs best on comparing images for pixel-level moving object detection. The score is determined by dividing the amount of true positives by the amount of actual moving objects. As seen in the image, the output of block 4 of SegNet performs best

With this ground-truth, we can test which stage of the SegNet network has the best features for detecting moving objects. To do this, we test the performance of the output of each block of three convolutional layers with 3×3 kernel size. This means we test 10 different feature maps. For every feature map, we obtain moving object detection results. To fairly compare the results from different feature maps, the threshold for each tested output layer is tuned until each layer returns roughly the same number of detected moving pixels. After that, we find how well the pixel-level moving object detection score compares with the ground-truth mask. This is done by computing the absolute difference between the masked area of the ground-truth dataset and the same area in the pixel-level moving object detection. By doing this for every sample in the ground-truth dataset, a score can be computed. This score is computed for every layer of interest. The result of this test is

shown in figure 4.8.

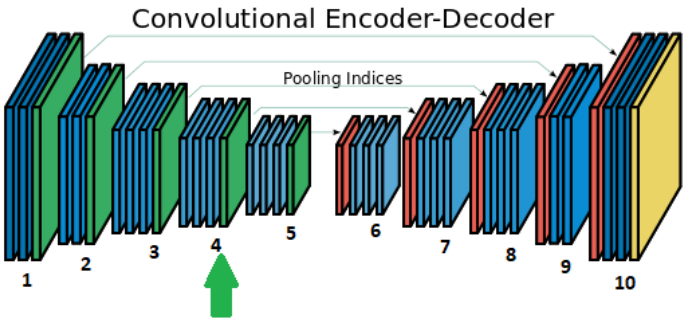


Figure 4.9: The SegNet architecture, with the output layers of which the feature maps are tested. The output which is used for the final implementation is the output of the fourth block, denoted with a green arrow.



Figure 4.10: Ground truth results of moving objects (left) compared to the pixel-level moving object detection results (right), using the feature maps of the fourth output block of SegNet. Note that for this visualization, only the pixels that are classified as moving with a full certainty is shown. When lower certainty detections are shown, locations of the moving objects in the other viewpoints are also visible, resulting in a bad visualization.

As can be seen in the image, the encoder part of the network performs significantly better than the decoder part of the network. This was expected, as the encoder part of the network gathers a dense description about the shapes, texture and colors of the input image. The decoder is more focused on turning this dense description into a class prediction. The best score was achieved by the output of the fourth block, which resulted in a score of 90.5%. The output of this layer is scaled down 8 times compared to the input image and has a 512 filters, which means an image consisting of four cube faces is described by a $64 \times 256 \times 512$ matrix. In figure 4.9, the different output layers of SegNet which are considered are shown. The layer which is used in the final implementation is marked with a green arrow.

This concludes the pixel-level moving object detection part of the pipeline. First, different approaches have been tested, and from those approaches SegNet was the most cost-efficient and accurate approach. After that, the different output layers of SegNet are compared to measure which layer gives the most effective feature maps. In the next section, the pixel-level moving object detection results are combined with the segmentation results. In figure 4.10, some pixel-level moving object detection results on ground-truth images are shown.

4.1.4. Combining of segmentation with pixel-level moving object detection

In the combine part of the pipeline, the result of the two branches are merged. As described in section 3.1.4, this is done by taking the segmentation result as a baseline and filtering the static objects out of the segmentation map, as seen in figure 3.8. Not much aspects of this part of the pipeline are experimentally decided. The most important experiment for this part of the pipeline is the value of the threshold. If the value is too high moving objects can be ignored but a too high value results in static objects or other false positives being inpainted. Unnecessarily inpainting static objects does not only reduce the quality of the image, it also slows down the pipeline as more inpainting has to be done. Because of the importance of the threshold value, a scientific approach is taken to experimentally find the best value. Every threshold value between zero and one is tested, with a 0.05 interval. With these values, moving object detection results are obtained for the images corresponding to the ground-truth moving object detection dataset. For every threshold the mean IoU is calculated. The threshold value versus the mean IoU is plotted in figure 4.11.

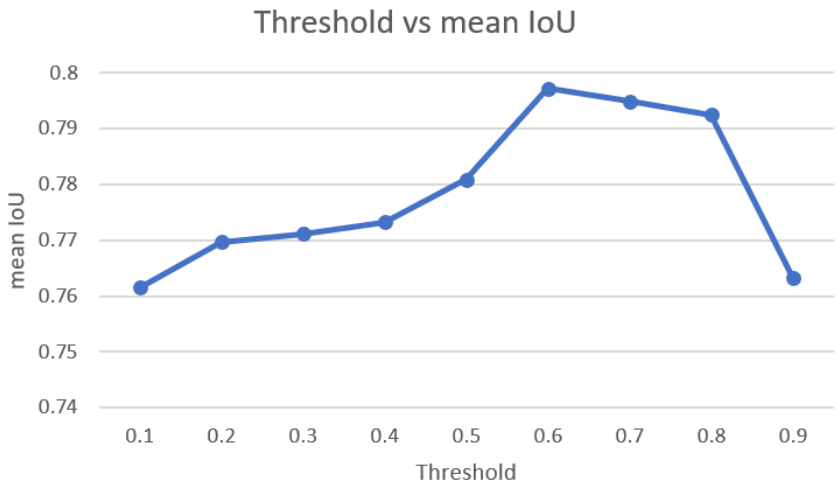


Figure 4.11: A plot of the mean IoU of the final moving object detection using different thresholds for the combination process. The score is measured on the ground-truth moving object detection dataset. The maximum mean IoU score is 0.80, achieved at a threshold of 0.6.

As seen in figure 4.11, the best mean IoU is achieved for a threshold of 0.6, resulting in a score of 0.80. This threshold is therefore used for the final pipeline. This finalizes the moving object detection part of the pipeline. Some visual results of the moving object detection part of the pipeline are shown in figure 4.12. The detections in those images are the detections which will be used for image inpainting. The way in which this image inpainting is done is explained in the next section.



(a)



(b)



(c)

Figure 4.12: Some results for the moving object detection part of the pipeline. The areas with an orange overlay are the areas in which moving objects are found by the algorithm. As seen in the images static objects which belong to classes of interest, like parked cars, are nicely filtered out of the segmentation result as they are lower than the threshold. The pipeline is good at detecting cars. Pedestrians are not always segmented correctly, as seen in image (a).

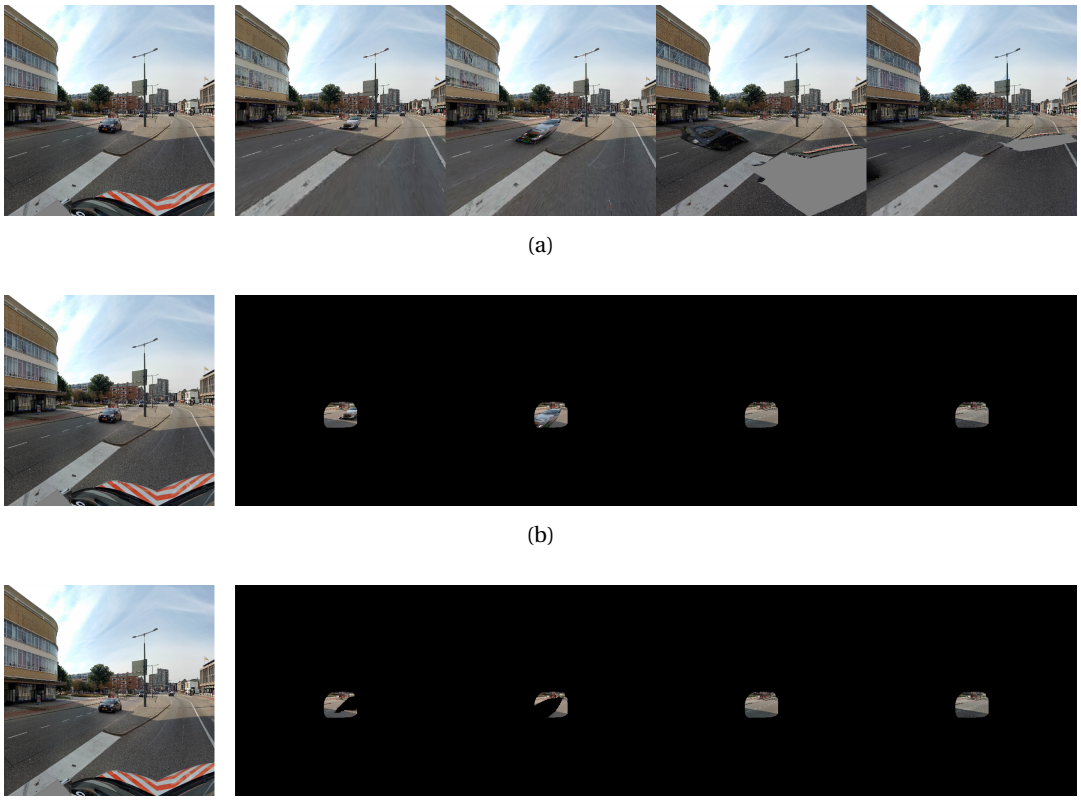
4.2. Image inpainting

Now the moving objects are found, they should be removed from the image data and replaced by a natural and realistic inpainting. This is done by the image inpainting part of the pipeline. Just like the combination of segmentation and pixel-level moving object detection in section 4.1.4, the image inpainting is also done by iterating over the objects and inpainting them one by one. As a quite detailed overview of the process of creating the input for the image inpainting network is already given in section 3.2.1 and 3.2.2, the experiments for this part are combined in section 4.2.1. In section 4.2.2 all details about the implementation and results of the inpainting network are presented.

4.2.1. Creating the input of the inpainting network

The first step of the inpainting process is to find which objects should be inpainted. This process is partly explained in section 3.1.4. The process consists of two steps. First, a binary mask of the possible moving objects is created using the moving object detection part of the pipeline. In the second step the separate contours in this binary mask are found. The contours are filtered, based on their size and spatial location. Every contour with an area below 50 pixels is removed to prevent overhead from running the inpainting network for irrelevant small objects. Objects with a center of gravity completely at the bottom of the image, near the recording car, and at the top of the image, in the sky, are also filtered out because no moving objects are usually present in this area. This result in the final group of contours of interest. For every contour of interest, an iteration of

the inpainting network is done to gather an inpainting for that specific shape. Because the network is trained to inpaint only one occlusion in an image, a mask specifically for this object is made, ignoring all the shapes of close-by other moving objects. A 512×512 patch selected around the center of this contour is selected, as the network is trained to inpaint 512×512 images. This patch, the mask, is one of the inputs for the inpainting network. A patch of the same area is extracted from the original input image and multiplied with the inverse of the mask. This results in a 512×512 image of the area around the moving object, without the moving object itself. This is the second input for the inpainting network. The other four input images are all taken from the re-projected images from other viewpoints.



(a)

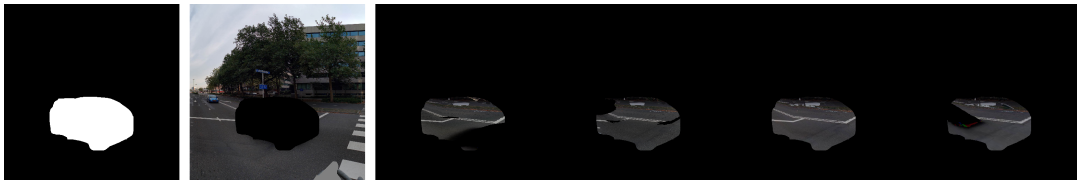
(b)

(c)

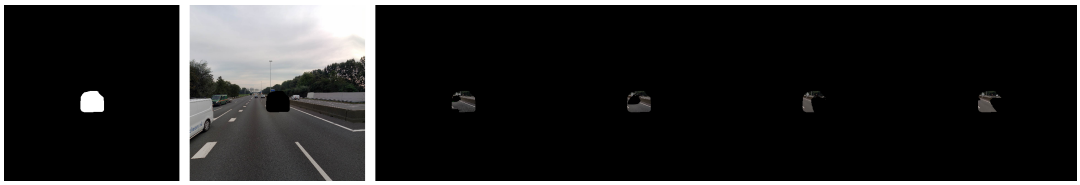
Figure 4.13: The different input types which where used for feeding information from other views to the inpainting network. On the left side you can see the image of interest with the car we want to remove from the image. On the right side you can see the information from other views. First, the four full re-projected images from other views were given as seen in (a). Later, all information outside of the occlusion was removed to reduce the amount of low-quality image data fed to the network (b). Finally, all parts of moving objects that are in the other view are removed to make sure the network only receives information of the actual background (c).

For the input images taken from the other viewpoints, several experiments have been done. For the first attempt, the patches corresponding to the patch from the input image were taken from the re-projected images and just simply fed to the network. This input can be seen in figure 4.13a. This way, basically all available information is given to the network. Training on this type of input data, however, did not give satisfying results. Results were slightly blurry and moving objects were often still slightly visible in the results. An explanation for the blurriness is the input data from other views. The quality of the re-projected images is not as high as the original images and feeding blurry input data to a network usually leads to blurry output. Therefore, a decision was made to reduce the amount of input data from the other view-points. First, all the data from other view-points outside of the shape of the moving object was removed. The goal of the network is to output an image which is exactly the same as the input image of interest, except for the occluded area for which an inpainting should be made. The part outside of the occluded area is best described by the input image of interest. This part, however, is given four more times by the input patches from different viewpoints. This information is not only redundant, but also blurry. Therefore, this information was removed from the input data as it could only lead to blurriness in the output. The result of this step can be seen in figure 4.13b.

Training on this data improves the blurriness, but leaves one more problem. The information from other view-points often still includes (parts of) re-projected moving objects. If a part of the occluded area is covered by other moving objects in one or more of the four re-projected images, the network has to learn from which of the four views it should select the information to get the best result. This process is learned in some cases but sometimes the output is blurry the color of the moving object which is in the other views is slightly visible in the output. Even though the networks does learn to pick the right information in a lot of cases, in some cases the moving objects are slightly visible which is a problem. The solution to this problem is to make sure other moving objects are not visible in the information from other viewpoints which means the network can't possibly select and use that information. To delete this information, the locations of all the moving objects in the re-projected images of the other views have to be known. This means that the full moving object detection process (getting a segmentation result and a pixel-level moving object detection and combining those) has to be performed for every separate view. This does not lead to a big increase in computational cost as most information is already available. The four separate pixel-level moving object detection results are already available because they are used to compute the final combined pixel-level moving object detection result. The images from other views have to be segmented and then re-projected to the right viewpoint, using the remap table that is already computed. The final step is to combine those two in the same manner as described in section 4.1.4. This results in four separate moving object detection results without a huge increase in computational cost. These four separate moving object detection results are used to remove the moving objects from the four input patches from other viewpoints, resulting in the final network input as seen in figure 4.13c.



(a)



(b)



(c)

Figure 4.14: Three examples of the full input data which is fed to the inpainting network to inpaint a single object. On the left side you can see the binary, 1 channel mask which marks the area that has to be inpainted. In the middle you can see the image of interest in which a moving object is removed, resulting in a black hole in the image which should be filled. On the right side you can see the information from the other views. These 6 images are concatenated and fed to the network as a $512 \times 512 \times 16$ matrix. The 16 channels consist of a 1-channel mask, a 3-channel image of interest and 4 3-channel images with information from other views.

This results in an input for the neural network consisting of a total of 6 images. One mask denoting the area that should be inpainted, the original image with a hole that has to be inpainted, and four images with information from the other views. This should give the network enough information to perform the inpainting. Some examples of the full final input of the image inpainting network are shown in figure 4.14. Now a network has to be trained which should be able to make an image inpainting based on this information. The way in which a dataset is made for training the neural network to make an inpainting based on this input

information is described in the next section. This section also explains the architecture of the network which is made for this purpose and results of the trained network.

4.2.2. The image inpainting network

In section 3.2.3, the architecture of the inpainting network was shown. This network has to be trained in order for it to perform well. As discussed before, a ground-truth is needed to train a neural network. This is a problem, because the ground truth is not known. We don't have an image taken at the exact same location without that moving object being there. Re-projected images are available, but have artifacts and a lower visual quality. Therefore, the network cannot be trained on actual removed objects. To train the network anyway, a synthetic dataset has been made in which we pretend that moving objects are somewhere in an image in which there actually is no moving object. This means we know the background of the imaginary moving object and can use that as a ground-truth. These imaginary moving objects can only be placed on parts of an image where no real moving object is present, otherwise the real ground-truth is still not known. For this thesis a piece of code was made to run the moving object detection part of the pipeline for a lot of images, find patches in these images in which there are no moving objects and use those locations to generate a sample for the synthetic dataset.

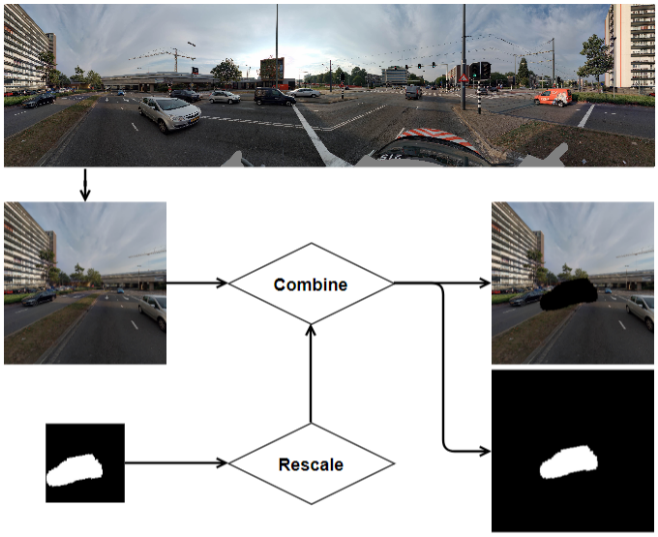


Figure 4.15: The way in which the synthetic dataset is created. From an image a patch is selected in which no moving object is in the middle, using the moving object detection algorithm. In this patch, an area is removed, which represents a deleted moving object. The patch with the occlusion, and the mask of the occlusion is used for the synthetic inpainting dataset. To complete the sample, information from other views is extracted using the moving object detection algorithm

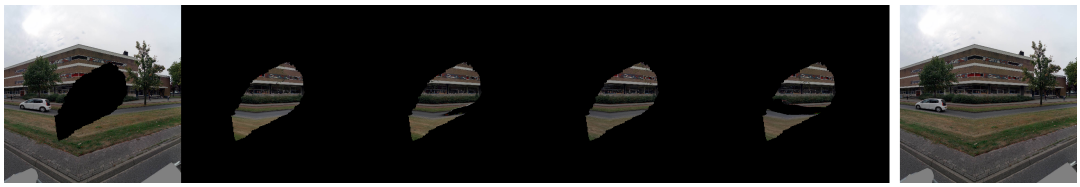
The process of creating this synthetic dataset is shown in figure 4.15. From a moving object detection result, a patch is selected in which there are no moving objects in the middle. In the middle of this patch, we remove an area with the shape of a moving object. The shapes are scaled to different scales, from 128×128 up to 384×384 to make sure the network is also trained on bigger and smaller moving objects. Even though the local discriminator only measures the quality of a 256×256 square, the networks still learns to inpaint successfully on a 384×384 occlusion. The shapes are used to occlude a small part of the image patch. Preferable, a patch just next to an actual moving object is occluded. This moving object will then occlude the area of interest in one or more of the re-projected neighbouring views. This way we can simulate the loss of information when the background of an actual occlusion is also occluded in some of the re-projected neighbouring views, a problem that will occur in the actual inpainting task in the pipeline.

Once the patch is selected and a random shape is removed the information from other point of views should be selected. A clear visualization of this process is given in figure 3.10. The removal of information from different views for the synthetic dataset is done in the same manner as in the actual inpainting pipeline, as described in section 4.2.1. All information from outside the occlusion is removed and all moving objects

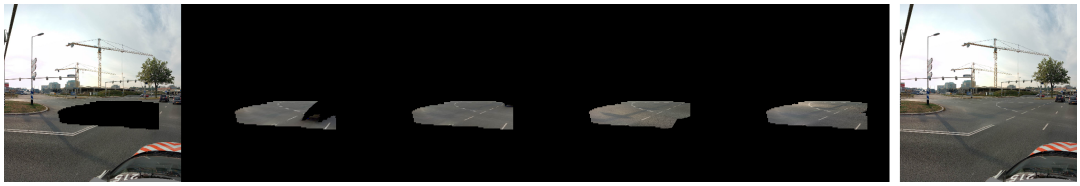
within the occluded area are removed. Three samples from the synthetic inpainting dataset are seen in figure 4.16. Those examples clearly show the difference in scale. The smallest one, figure 4.19a, simulates moving objects which are small or at a far distance. The largest one, figure 4.19d, simulates moving objects at a close range. The first five tiles of the data samples show the input data which consist of the image of interest with the removed occlusions, and the four images with information from different views. The last tile is the ground truth data, which is used for the discriminator.



(a)



(b)



(c)

Figure 4.16: A few samples from the synthetic dataset. On the left side you can see the image in which we pretend a moving object is removed, together with the information from the four different views which is used as input for the network. As there were no actual moving objects in the picture, the ground-truth information of the background is available, which is seen on the right side. The network is trained on occlusions from different scales ranging from 128×128 to 384×384, to make the network robust to moving objects at a close range and moving objects further away.

The above mentioned process is repeated till the dataset for both training and evaluation is ready. As mentioned in section 4.2.1, the training setup was changed twice. First the network was given the full re-projected views, then without the area around the occlusion and lastly without the moving objects. A new type of input also means a new dataset has to be created. This means several datasets were used in this thesis. When a new dataset was finished, the training didn't start from scratch again. The final weights from the training on the previous dataset were used to start training on the new dataset. This speeds up the training significantly compared to starting the training from scratch again. The final training dataset, without the moving objects in the other views, consist of 7000 samples. As the network is already trained for a long time on the previous synthetic datasets, not much training is required on this final dataset. The weights only need to be fine-tuned on the slightly different type of data. The network is fine-tuned on the final synthetic dataset for 50k iterations. The performance of the network is measured using the evaluation dataset which consists of 1000 inpainting samples. The average L1 loss and PSNR on this evaluation dataset is calculated. The L1 loss is 2.51% and the PSNR is 27.19. These scores are significantly better than the inpainting scores of the state-of-the-art networks. [37][36][33][35]. A comparison with the state-of-the-art works is shown in table 4.2. As no major improvements are made to the network itself, this does proof that feeding information from different views does significantly improve the image inpainting. Some of the final inpainting results and the corresponding ground-truth images are seen in figure 4.17. These images belong to the evaluation dataset. The network never saw these images while learning to inpaint, which means it did not overfit on those samples.

Authors	L1 loss	PSNR
Song et al.	15.61%	-
Pathak et al.	10.33%	17.59 dB
Zhao et al.	-	18.41 dB
Yu et al.	8.60%	18.91 dB
Ours	**2.51%**	**27.19 dB**

Table 4.2: A comparison of the performance of our inpainting network with the state-of-the-art in Generative Adversarial Networks for inpainting. As no major improvements are made to the networks, this shows that using information from multiple views has a significant influence on image inpainting.



Figure 4.17: In-painting results on the test set. These samples were not seen by the network during training. Except from some fine details, the inpainting is not distinguishable from the ground-truth. Note the wheel of the car in the top picture, and the artifact at the border of the inpainting in the middle second output picture.

The weights after 50k iterations are stored. This checkpoint is used for the implementation of the inpainting network in the pipeline. The first part of this implementation was already described in figure 3.10. All separate moving objects are found and filtered, and for each of those all six input tiles for the network are computed. Then for each of these moving objects an iteration of the network is performed. From each output of the network, the part within the contour of the moving object is selected. This part is used to replace the moving object in the original image. After repeating this process for all contours, the image should be free of any moving objects.

This last step finishes the pipeline. The final goal of this thesis, the original image without any moving objects in it, is returned. Some examples of the inpainted result can be seen in figure 4.19. This final part of the pipeline is hard to quantify, as no ground-truth information is available for this output data. Instead of quantifying the final result, the separate parts of this pipeline are all quantified with their own ground-truth datasets and metrics. As no good measurements can be done to quantify the final results of this thesis, a survey is performed to give a good indication of the final results.

This survey was performed on employees of CycloMedia which are familiar with the type of images shown in the survey. These people can be considered experts on the data and therefore should perform well on finding disruptions in the images. A total of 30 image tiles were shown to these participants. In 15 of these 30 image tiles, a moving object was present which was removed by the pipeline. These 15 results were selected randomly from a large number of results, but bad results caused by problems in the LIDAR processing pipeline were filtered out as these errors are out of the scope of this thesis. The number of tiles in which a moving object has been removed was not revealed to the participants.

Each participant was asked to look at the image tiles for approximately 10 seconds for each. In these 10 seconds, the participant has to determine if a moving object has been removed from the tile. Participants were told to pay close attention to misplaced shadows, blurry areas and other artifacts. After each tile, the participant has to answer a yes/no question, asking if a moving object was removed from the image. For the validity of this survey, the correct answers were not shown to the participants before everyone filled in the survey. This means participants could not influence the scores of others by spreading the answers around.

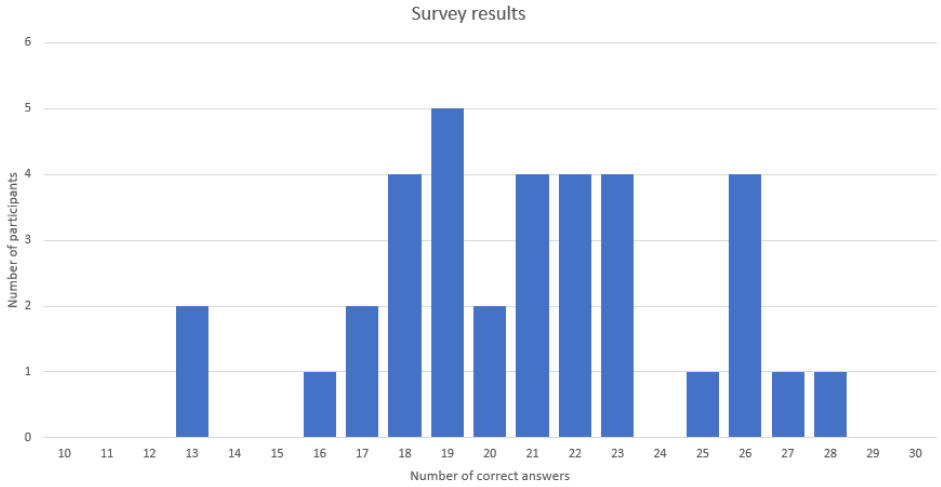


Figure 4.18: A histogram showing the results of the survey in which employees of CycloMedia are asked in which images a moving object has been removed. On the x-axis the scores are shown. On the y-axis the number of participants which achieved each score.

A total of 35 employees of CycloMedia participated in the survey. All participants answered the 30 yes/no questions and got 1 point for each correct answer. The average score over all participants is 20.89, with a median of 21 out of 30 points. A full overview of the distribution of the final scores can be seen in figure 4.18. As the questions are given in a yes/no manner, an average score of 15 can be reached by randomly answering the questions. The remaining 15 points can be achieved by actually noticing the difference between ground-truth data and output of the pipeline. With an average score of 20.89, it becomes clear that participants can recognize inpainted areas in some cases but often fail to find the inpainted tiles. In some cases, the inpainted area can be recognized by for instance a shadow, which is not considered a harmful artifact. This can also be seen in the survey results, where tiles in which a clear shadow remains are recognized as an in-painted tile by the majority of the participants. This means that the amount of inpainted areas that are recognized by actual artifacts is very low, meaning the pipeline performs well. All tiles which were shown to the participants, and the ground-truth images, can be seen in appendix A. This appendix also shows the number of yes and no votes for each separate question.

With the results of this survey, the experiments and results section is finished. For every separate part of the pipeline, the experiments which led to the final pipeline have been thoroughly explained. The separate parts have been quantified, each with their own metric. Finally, the quality of the full pipeline is measured with a survey. A larger number of output images of the pipeline can be seen in appendix B. In appendix C, results on a sequence of recordings is shown.



(a)



(b)



(c)



(d)

Figure 4.19: Two examples of results from the image inpainting part of the pipeline. Image (a) and (c) are outputs of the moving object detection pipeline. Image (b) and (d) are the results of feeding image (a) and (c) to the image inpainting pipeline.

5

Discussion

In this thesis, a fully automated pipeline is created which is able to detect and remove moving objects in street-view imagery. In chapter 3, the higher level overview of this pipeline is presented and the detailed design of every part of the pipeline is explained. In chapter 4, all experiments that led to this final design and the results of the final pipeline are presented. In this section, the results are discussed. The contributions, strengths and limitations of the pipeline as well as ideas for possible future research are discussed.

The first step of the pipeline is the computation of the pixel-level moving object detection. The best results were achieved by computing a feature map of the image of interest and re-projecting the nearby views. By using SegNet features for this comparison, an average of 90.5% of the pixels that belong to moving objects were classified as moving. Using only this method, however, gave two problems. The inaccuracies and errors in the LIDAR data leads to imperfect re-projections, which leads to a significant amount of false positives in the pixel level moving object detection. Next to that, pixel-level results do not give an accurate outline of the moving objects, which means the pixel-level results should be grouped by a different approach. A second approach is necessary.

This second approach is the segmentation network. The goal of this network is to find the mask of all objects of interest. After training a custom version of SegNet[11] on the annotated CycloMedia dataset for segmentation, a mean IoU of 0.583 is reached. This score is lower than the state-of-the-art scores which are achieved on publicly available annotated street-view imagery datasets like Mapillary Vistas [39] and the cityscapes dataset [38]. This has two main causes. First, the used network. SegNet was published three years ago, but because the research field of neural networks is rapidly improving, many current networks outperform SegNet. The reason for choosing SegNet is the low computational cost, which speeds up the pipeline, and the simplicity of the network architecture. Current state-of-the-art segmentation network often consist of several sub-networks or branches. As the segmentation network should also be used to extract a feature map description of images (for the pixel-level detection), it is desirable to have a straightforward network without information passing through different branches. The second reason is the dataset used. This dataset, with 2800 training and 1200 evaluation samples, is relatively small compared to the public datasets. This mainly effects the scores of pedestrians and bicycles, which are very scarce in the dataset.

Another problem in the segmentation part of the pipeline are objects of interest that overlap each other. A row of parked cars in an image could be segmented as one big object. This gets even more problematic if a moving car is driving in front of a row of parked cars. This results in one big detection which includes both moving and static objects. Depending on the comparison with the pixel-level moving object detection, this either results in static objects being incorrectly removed in the images or moving objects being incorrectly retained. For future work, instance segmentation could be used to solve this problem. In instance segmentation, the network does not only learn to distinguish between classes but also finds the separate instances of the class. This means that, for a row of parked cars, separate masks for every car are returned. Another suggestion for future work is to first train the segmentation network on a big public dataset of annotated street-view imagery, with a greater presence of pedestrians and cyclists. This way the network gets a better understanding for those classes. After that, the network could be fine-tuned on the CycloMedia dataset to

improve performance. By pre-training the model, the network might perform better in the detection of the classes that are scarce in the CycloMedia dataset.

To combine the segmentation and pixel-level moving object detection branches in the pipeline, a simple but effective approach is used. In this approach, the segmentation result is used as a baseline. The only tuneable parameter of this part of the pipeline is the threshold, for which the value is determined in a scientific manner using the mean IoU. The maximum mean IoU was reached at a threshold of 0.6, reaching a score of 0.797. Overall, the moving object detection pipeline shows good results, which could be improved even more by an improved segmentation network.

The inpainting part of the pipeline is a unique approach, in which we combined information from different points of view with Generative Adversarial Networks. Because this is a new approach and no public datasets are available on this specific inpainting task, it is harder to judge the quality of this network. The only way to assess the quality of the inpainting network is the score on the evaluation dataset, which is measured using three metrics. These are the L1 loss and Peak Signal to Noise Ratio (PSNR). Scores of 2.51% (L1) and 27.19 (PSNR) are reached. This is significantly better than comparable inpainting networks which don't use information from other points of view[33][34][35][36][37], which indicates using information from other points of view results in a big performance boost for inpainting networks.

The inpainting network generally performs well if sufficient background information is available. Poor results occur sometimes when a vehicle is driving in the same lane as the recording vehicle. Especially when the moving object is farther away, big parts of the background of the moving object can be occluded in every other point of view. This problem does not occur in the synthetic dataset and therefore the network has a bad performance in this situation. This problem also occurs when an object is static but not in the LIDAR point-cloud. This often happens for cars at a traffic light or parked cars in a street which is recorded multiple times. In this case, the actual background is also not visible in the other views. Bad results resulting from those two problems can be seen in appendix B. For future work, a possible solution to this problem is to remove big parts of the information from other views in a part of the synthetic dataset. By doing this, the network will also learn to inpaint when almost no information from other views is available and therefore be more robust in those situations. Next to removing information from other views, results could still be slightly improved by increasing and improving the dataset. A bigger dataset with a higher variety and better balanced classes will improve the performance.

Each part of this thesis is quantified with ground-truth data or evaluation datasets, using standard evaluation metrics. This results in a full connected pipeline with a high validity, of which every aspect is separately tested. Because for the output of the pipeline no ground truth is available, the final result is tested with a survey. This survey shows that, assuming the LIDAR point-cloud has no major errors, the pipeline in-paints images in such a way that finding the inpainted areas is a difficult tasks for people that are experienced with the type of data.

6

Conclusion

The goal of this thesis was to find out how the available data at CycloMedia can be used in an efficient way to generate a pipeline which finds and removes moving objects in street-view imagery and generates a natural inpainting which consists of the actual information behind the moving objects. The final pipeline can be divided into two parts, in which the first part finds moving objects (motorized vehicles, pedestrians, bicycles) in the street-view imagery and the second part removes those found objects from the images and generates an inpainting of the background using information from other point of views.

For the moving object detection part of the pipeline, image data from different point of views, LIDAR point clouds and an annotated segmentation dataset was available. The large baseline between the images and the lack of moving objects in the LIDAR point-clouds made most state-of-the-art moving object detection pipelines unavailable for this thesis. A pipeline with two branches has been designed which combines image segmentation with a pixel-level moving object detection approach.

The first branch of the moving object detection pipeline, the image segmentation part, uses neural networks and the dataset of CycloMedia images in which all objects of interest are carefully annotated. An altered version of SegNet, which has been updated to newer standards in the state-of-the-art in machine learning, is trained on this dataset until the best performance on the evaluation dataset is found. This results in a mask of all objects of interest in the image. This method, however, does not distinguish between moving and static objects. Therefore, a second approach is used to determine which objects are moving.

For this approach, the pixel-level moving object detection, images taken within close range of the image of interest are re-projected and visualized as if they where taken from the point of view of the image of interest. The re-projected images are compared with the image of interest using a feature-map based comparison technique. Extensive tests have been done to find the correct network and output layer resulting in a pixel-level moving object detection with a high accuracy.

At the end of the pipeline, both branches are combined to find the moving objects. For the objects found in the segmentation branch, a distinction is made between the static and the moving objects. This is done by overlapping the separate objects in the segmentation mask with the pixel-level moving object detection result. The overlap between those two are computed and compared to a threshold which has been determined in an experimental way. If the overlap is higher than the threshold, the object is classified as moving. If the overlap is lower, the object is classified as static and not considered in the image inpainting part of the pipeline.

All separate parts of the pipeline has been extensively tested, and the final output of the moving object detection part of the pipeline shows good results. Based on the separate tests, the most further improvements can be gained in the segmentation branch of the pipeline. The outlines of masks are not always perfect and the network does not always find pedestrians and bicycles. If this part can be improved even more, the combination with the already accurate pixel-level moving object detection method will result in a even better

pipeline with highly accurate moving object detection results.

For the image inpainting part of the pipeline the aim is to replace the moving objects by their correct background without introducing too much artifacts to the images. To do this, a novel method has been introduced which combines Generative Adversarial Networks (GANs) and information from other point of views into a multi-view neural network inpainting approach. This is different from the state-of-the-art as, to my best knowledge, no other publication combines GANs with information from multiple views.

Extensive experiments have been done to determine how the information from other views should be fed to the network. The conclusion of these experiments was that, as the re-projected images are are blurry, only the most relevant information from other views should be used for the inpainting. The area around the occlusion should stay exactly the same as the input image of interest. Therefore, this area should not be fed to the network in the information from other point of views, as this information is redundant and has a lower image quality than the image of interest. Besides that, moving objects that appear in the area of interest in the other point of views are also removed. This information should not be included in the final inpainting and therefore feeding it to the network only complicates the image inpainting task.

The network which is designed to use this input and create an image inpainting is a fully convolutional network with an encoder-decoder structure. Between the encoder and decoder, convolutional layers with dilated convolution are used to interchange information between different spatial locations of the image. To train the network, a local and a global discriminator are used to compare the output of the generator with the ground truth. The local discriminator measures how correct the inpainted area is and the global discriminator analyzes the whole image and how well the inpainting fits into the full image. The network is trained on a synthetic dataset which is made specifically this thesis. After training has converged, the weights of the network are used for the inpainting part of the full pipeline. This implementation iterates over all the moving objects which are found by the first part of the pipeline and for every object it generates an inpainting and replace the moving object with the inpainting.

These two parts have been combined in a fully automated pipeline. For each image of interest, the pipeline finds the previous and next two images in the sequence, uses this information to find the moving objects, and replaces the objects with a realistic inpainting. Furthermore, an approach is suggested to combine image segmentation with a LIDAR based method to distinguish between static and moving objects. Lastly, a novel method for image inpainting is introduced which combines a Generative Adversarial Network with information from different point of views.

Bibliography

[1] Heuvel, F. V., Beers, B., & Verwaal, R. (2007). U.S. Patent No. 7961979. Washington, DC: U.S. Patent and Trademark Office.

[2] Krizhevsky, A., Sutskever, I., & Hinton, G. E. (2012). Imagenet classification with deep convolutional neural networks. *In Advances in neural information processing systems* (pp. 1097-1105).

[3] Simonyan, K., & Zisserman, A. (2014). Very deep convolutional networks for large-scale image recognition. *International Conference on Learning Representations*

[4] He, K., Zhang, X., Ren, S., & Sun, J. (2016). Deep residual learning for image recognition. *In Computer Vision and Pattern Recognition* (pp. 770-778).

[5] Girshick, R. (2015, December). Fast R-CNN. *In International Conference on Computer Vision* (pp. 1440-1448).

[6] Ren, S., He, K., Girshick, R., & Sun, J. (2015). Faster R-CNN: Towards real-time object detection with region proposal networks. *In Advances in Neural Information Processing Systems* (pp. 91-99).

[7] Dai, J., Li, Y., He, K., & Sun, J. (2016). R-FCN: Object detection via region-based fully convolutional networks. *In Advances in Neural Information Processing Systems* (pp. 379-387).

[8] Liu, W., Anguelov, D., Erhan, D., Szegedy, C., Reed, S., Fu, C. Y., & Berg, A. C. (2016). Ssd: Single shot multibox detector. *In European Conference on Computer Vision* (pp. 21-37).

[9] He, K., Gkioxari, G., Dollár, P., & Girshick, R. (2017). Mask r-cnn. *In International Conference on Computer Vision* (pp. 2980-2988).

[10] Long, J., Shelhamer, E., & Darrell, T. (2015). Fully convolutional networks for semantic segmentation. *In Computer Vision and Pattern Recognition* (pp. 3431-3440).

[11] Badrinarayanan, V., Kendall, A., & Cipolla, R. (2015). SegNet: A deep convolutional encoder-decoder architecture for image segmentation. *In Pattern Analysis and Machine Intelligence 39(12)* pp 2481-2495.

[12] Zhao, H., Shi, J., Qi, X., Wang, X., & Jia, J. (2017). Pyramid scene parsing network. *In Computer Vision and Pattern Recognition* (pp. 2881-2890).

[13] Chen, L. C., Papandreou, G., Schroff, F., & Adam, H. (2017). Rethinking atrous convolution for semantic image segmentation. *arXiv preprint arXiv:1706.05587*.

[14] Sevilla-Lara, L., Sun, D., Jampani, V., & Black, M. J. (2016). Optical flow with semantic segmentation and localized layers. *In Computer Vision and Pattern Recognition* (pp. 3889-3898).

[15] Sun, D., Wulff, J., Sudderth, E. B., Pfister, H., & Black, M. J. (2013). A fully-connected layered model of foreground and background flow. *In Computer Vision and Pattern Recognition* (pp. 2451-2458).

[16] Lenz, P., Ziegler, J., Geiger, A., & Roser, M. (2011). Sparse scene flow segmentation for moving object detection in urban environments. *In Intelligent Vehicles Symposium* (pp. 926-932).

[17] Menze, M., & Geiger, A. (2015). Object scene flow for autonomous vehicles. *In Computer Vision and Pattern Recognition* (pp. 3061-3070).

[18] Enzweiler, M., Hummel, M., Pfeiffer, D., & Franke, U. (2012). Efficient stixel-based object recognition. *In Intelligent Vehicles Symposium* (pp. 1066-1071).

[19] Erbs, F., Schwarz, B., & Franke, U. (2013). From stixels to objects—A conditional random field based approach. *In Intelligent Vehicles Symposium* (pp. 586-591).

[20] Schneider, L., Cordts, M., Rehfeld, T., Pfeiffer, D., Enzweiler, M., Franke, U., ... & Roth, S. (2016). Semantic stixels: Depth is not enough. *In Intelligent Vehicles Symposium* (pp. 110-117).

[21] Azim, A., & Aycard, O. (2012). Detection, classification and tracking of moving objects in a 3D environment. *In Intelligent Vehicles Symposium* (pp. 802-807)

[22] Ferri, F., Gianni, M., Menna, M., & Pirri, F. (2015, September). Dynamic obstacles detection and 3d map updating. *In International Conference on Intelligent Robots and Systems* (pp. 5694-5699)

[23] Cho, H., Seo, Y. W., Kumar, B. V., & Rajkumar, R. R. (2014). A multi-sensor fusion system for moving object detection and tracking in urban driving environments. *In International Conference on Robotics and Automation* (pp. 1836-1843).

[24] Yan, J., Chen, D., Myeong, H., Shiratori, T., & Ma, Y. (2014). Automatic extraction of moving objects from image and LIDAR sequences. *In 3D Vision* (Vol. 1, pp. 673-680)

[25] Marković, I., Chaumette, F., & Petrović, I. (2014). Moving object detection, tracking and following using an omnidirectional camera on a mobile robot. *In International Conference on Robotics and Automation* (pp. 5630-5635)

[26] Criminisi, A., Pérez, P., & Toyama, K. (2004). Region filling and object removal by exemplar-based image inpainting. *In IEEE Transactions on image processing, 13(9),* (pp. 1200-1212).

[27] Barnes, C., Shechtman, E., Finkelstein, A., & Goldman, D. B. (2009). PatchMatch: A randomized correspondence algorithm for structural image editing. *In ACM Transactions on Graphics, 28(3),* (pp. 24).

[28] Buyssens, P., Daisy, M., Tschumperlé, D., & Lézoray, O. (2015). Exemplar-based inpainting: Technical review and new heuristics for better geometric reconstructions. *In IEEE Transactions on image processing, 24(6),* (pp. 1809-1824).

[29] Hervieux, A., Papadakis, N., Bugeau, A., Gargallo, P., & Caselles, V. (2011). Stereoscopic image inpainting using scene geometry. *In International Conference on Multimedia and Expo* (pp. 1-6).

[30] Flores, A., & Belongie, S. (2010). Removing pedestrians from google street view images. *In Computer Vision and Pattern Recognition Workshops* (pp. 53-58).

[31] Thaskani, S., Karande, S., & Lodha, S. (2015). Multi-view image inpainting with sparse representations. *In International Conference on Image Processing* (pp. 1414-1418).

[32] Thonat, T., Shechtman, E., Paris, S., & Drettakis, G. (2016). Multi-view inpainting for image-based scene editing and rendering. *In International Conference 3D Vision* (pp. 351-359).

[33] Pathak, D., Krahenbuhl, P., Donahue, J., Darrell, T., & Efros, A. A. (2016). Context encoders: Feature learning by inpainting. *In Computer Vision and Pattern Recognition* (pp. 2536-2544)

[34] Iizuka, S., Simo-Serra, E., & Ishikawa, H. (2017). Globally and locally consistent image completion.*In ACM Transactions on Graphics, 36(4),* (pp. 107).

[35] Zhao, G., Liu, J., Jiang, J., & Wang, W. (2017). A deep cascade of neural networks for image inpainting, deblurring and denoising. *In Multimedia Tools and Applications,* (pp. 1-16).

[36] Song, Y., Yang, C., Lin, Z., Li, H., Huang, Q., & Kuo, C. C. J. (2017). Image inpainting using multi-scale feature image translation. *arXiv preprint arXiv:1711.08590.*

[37] Yu, J., Lin, Z., Yang, J., Shen, X., Lu, X., & Huang, T. S. (2018). Generative image inpainting with contextual attention.

[38] M. Cordts, M. Omran, S. Ramos, T. Rehfeld, M. Enzweiler, R. Benenson, U. Franke, S. Roth, & B. Schiele (2016). The Cityscapes Dataset for Semantic Urban Scene Understanding. *In Computer Vision and Pattern Recognition*

[39] Neuhold, G., Ollmann, T., Bulò, S. R., & Kontschieder, P. (2017). The Mapillary Vistas Dataset for Semantic Understanding of Street Scenes. *In International Conference on Computer Vision* (pp. 5000-5009).

[40] Olah, et al., Feature Visualization, Distill, 2017.

[41] Russakovsky, O., Deng, J., Su, H., Krause, J., Satheesh, S., Ma, S., Huang, Z., Karpathy, A., Khosla, A., Bernstein, M., Berg, A.C., Fei-Fei, L. ImageNet *Large Scale Visual Recognition Challenge*. arXiv:1409.0575, 2014

[42] Arandjelovic, R., Gronat, P., Torii, A., Pajdla, T., & Sivic, J. (2016). NetVLAD: CNN architecture for weakly supervised place recognition. *In Computer Vision and Pattern Recognition* (pp. 5297-5307).

[43] Minaee, S., Abdolrashidiy, A., & Wang, Y. (20167). An experimental study of deep convolutional features for iris recognition. *In Signal Processing in Medicine and Biology Symposium* (pp. 1-6)

A

Survey questions

A.1. Correct tiles - Tiles in which a moving object was removed



(a) **Yes: 19 (54.3%)** No: 16 (45.7%)



(b) **Yes: 16 (45.7%)** No: 19 (54.3%)



(c) **Yes: 21 (60.0%)** No: 14 (40.0%)



(d) **Yes: 8 (22.9%)** No: 27 (77.1%)



(e) **Yes: 20 (57.1%)** No: 15 (42.9%)



(f) **Yes: 28 (80.0%)** No: 7 (20.0%)



(g) **Yes: 33 (94.3%)** No: 2 (5.7%)



(h) **Yes: 22 (62.9%)** No: 13 (37.1%)



(i) **Yes: 21 (60.0%)** No: 14 (40.0%)



(j) **Yes: 34 (97.1%)** No: 1 (2.9%)



(k) **Yes: 7 (20.0%)** No: 28 (80.0%)



(l) **Yes: 29 (82.9%)** No: 6 (17.1%)



(m) **Yes: 30 (85.7%)** No: 5 (14.3%)



(n) **Yes: 29 (82.9%)** No: 6 (17.1%)




(o) **Yes: 16 (45.7%)** No: 19 (54.3%)

A.2. Incorrect tiles - Tiles in which no moving object was removed



(a) Yes: 17 (48.6%) **No: 18 (51.4%)**



(b) Yes: 8 (22.9%) **No: 27 (77.1%)**



(c) Yes: 14 (40.0%) **No: 21 (60.0%)**



(d) Yes: 8 (22.9%) **No: 27 (77.1%)**



(e) Yes: 12 (34.3%) **No: 23 (65.7%)**



(f) Yes: 10 (28.6%) **No: 25 (71.4%)**



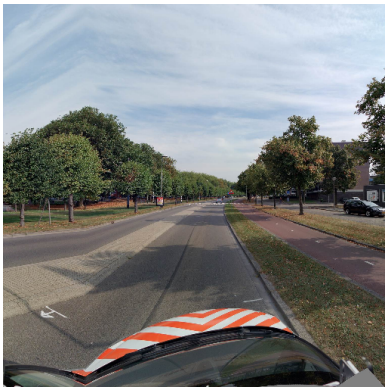
(g) Yes: 6 (17.1%) **No: 29 (82.9%)**



(h) Yes: 17 (48.6%) **No: 18 (51.4%)**



(i) Yes: 5 (14.3%) **No: 30 (85.7%)**



(j) Yes: 2 (5.7%) **No: 33 (94.3%)**



(k) Yes: 7 (20.0%) **No: 28 (80.0%)**



(l) Yes: 8 (22.9%) **No: 27 (77.1%)**



(m) Yes: 5 (14.3%) **No: 30 (85.7%)**



(n) Yes: 2 (5.7%) **No: 33 (94.3%)**



(o) Yes: 6 (17.1%) **No: 29 (82.9%)**

B

Various pipeline outputs

In this appendix the input and output of the pipeline for a variety of scenes is shown. To show that this approach works in different environment, both rural and urban scenes are shown. Besides that, the images have a wide variety of moving objects to show that the approach works for different types of moving objects. In the second part of this appendix, some problem cases are shown. For each scene, two images are shown. The first image is the original image which is fed to the pipeline. The second image is the final output of the pipeline in which the moving objects are removed.



(a)



(b)



(c)



(d)



(e)



(f)

Figure B.1: A number of results in various locations in which different types of moving objects are removed to show the variety of scenes in which this method is applicable.



(a) Example of errors occurring for cars that drive in the same line as the recording vehicle. For these cars, parts of the background is not visible in any of the other views.



(b) Another example of errors occurring because the background is not visible in any other view. In this case, the cars are static but because they don't appear in the LIDAR point-cloud, they are classified as moving objects.



(c) An example of artifacts that occur when the segmentation is not good enough. Only the segmented part, with a buffer area around it, is removed and inpainted. This sometimes results in remaining parts of moving objects in images.

C

Sequences of Cycloramas

In this appendix the halfway and final output of the pipeline for two small sequences is shown. The aim of this section is to proof that it works on consecutive images in various scenes. Two scenes are shown. The first one is a highway with a several cars. The other one is a roundabout with cars, cyclists and scooters.



(a)



(b)





(c)





(d)





(e)





(f)



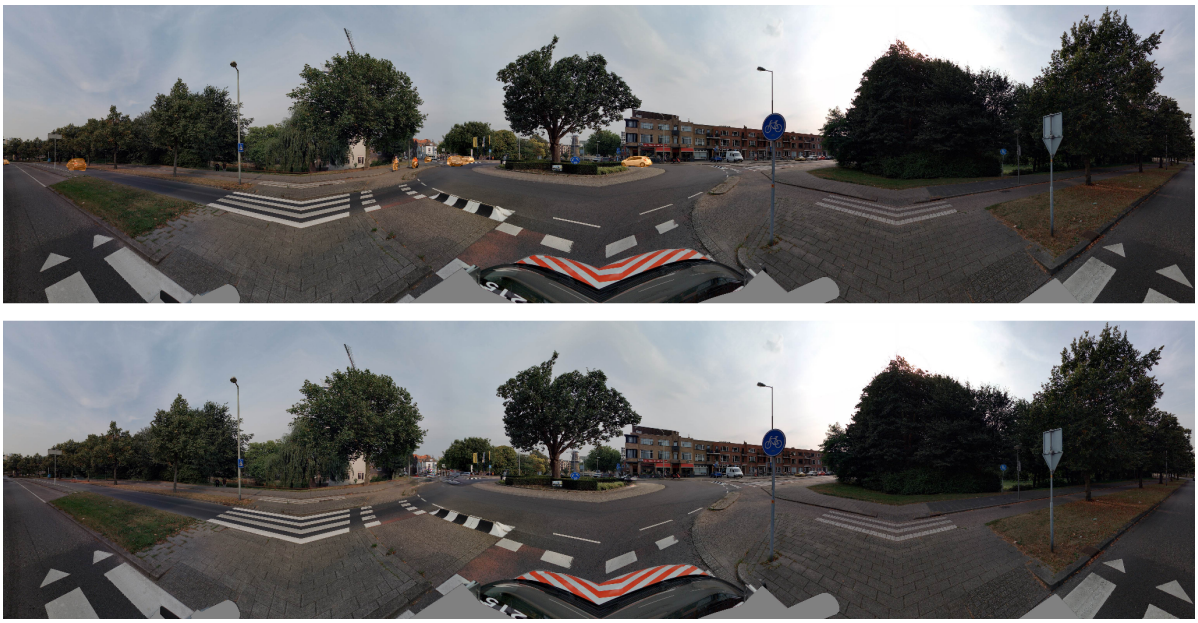


(g)

Figure C.1: Sequence results in highway traffic



(a)



(b)



(c)





(d)





(e)

Figure C.2: Sequence results in urban traffic