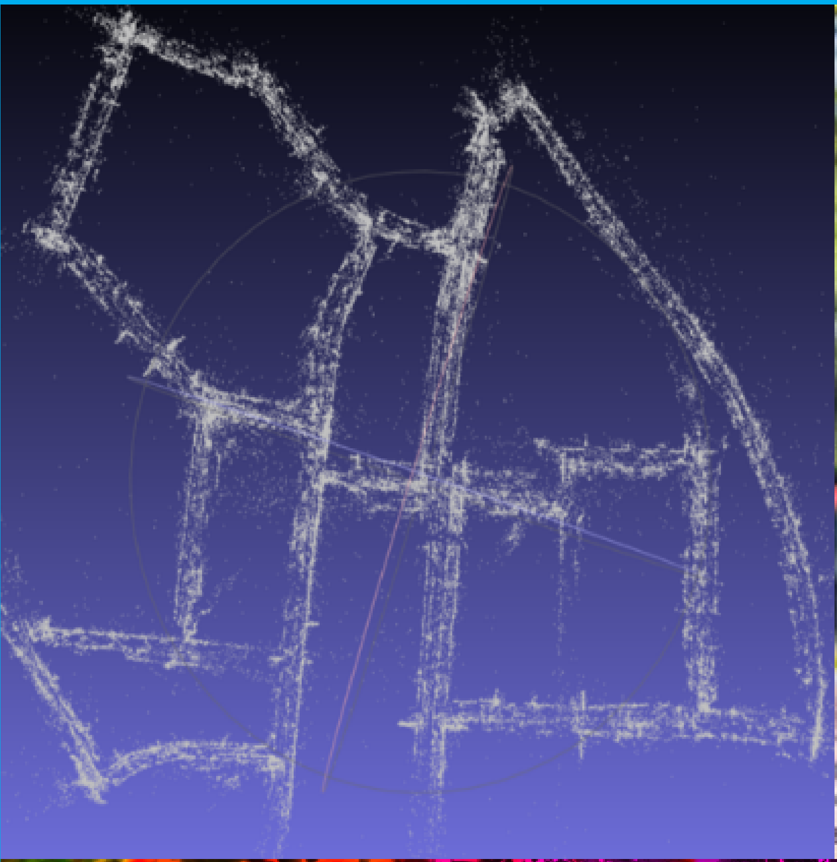
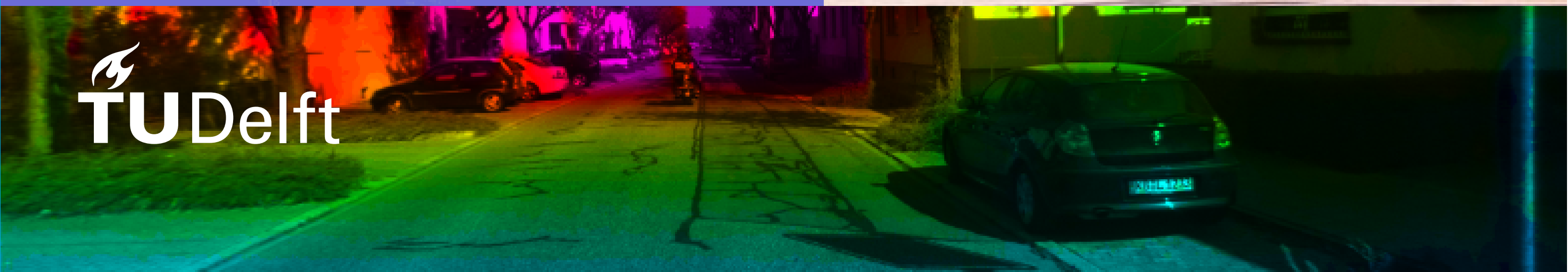
Improving Monocular SLAM

using Depth Estimating CNN

J. T. Zijlmans





Improving Monocular SLAM

using Depth Estimating CNN

by

J. T. Zijlmans

to obtain the degree of Master of Science
at the Delft University of Technology,
to be defended publicly on Friday July 6, 2018 at 11:30 AM.

Student number:	4221249
Date:	June 22, 2018
TU Delft Supervisor:	Dr. ir. J. F. P. Kooij
RCS Supervisors:	Ir. F. Gaisser & Dr. F. K. Nejadasl
Committee members:	Prof. dr. ir. P. P. Jonker & Dr. A. Tejada Ruiz

Abstract

To bring down the number of traffic accidents and increase people's mobility companies, such as Robot Engineering Systems (RES) try to put automated vehicles on the road. RES is developing the WEpod, a shuttle capable of autonomously navigating through mixed traffic. This research has been done in cooperation with RES to improve the localization capabilities of the WEpod.

The WEpod currently localizes using its GPS and lidar sensors. These have proven to be not accurate and reliable enough to safely navigate through traffic. Therefore, other methods of localization and mapping have been investigated. The primary method investigated in this research is monocular Simultaneous Localization and Mapping (SLAM). Based on literature and practical studies, ORB-SLAM has been chosen as the implementation of SLAM.

Unfortunately, ORB-SLAM is unable to initialize the setup when applied on WEpod images. Literature has shown that this problem can be solved by adding depth information to the inputs of ORB-SLAM. Obtaining depth information for the WEpod images is not an arbitrary task. The sensors on the WEpod are not capable of creating the required dense depth-maps. A Convolutional Neural Network (CNN) could be used to create the depth-maps.

This research investigates whether adding a depth-estimating CNN solves this initialization problem and increases the tracking accuracy of monocular ORB-SLAM. A well performing CNN is chosen and combined with ORB-SLAM. Images pass through the depth estimating CNN to obtain depth-maps. These depth-maps together with the original images are used in ORB-SLAM, keeping the whole setup monocular.

ORB-SLAM with the CNN is first tested on the Kitti dataset. The Kitti dataset is used since monocular ORB-SLAM initializes on Kitti images and ground-truth depth-maps can be obtained for Kitti images. Monocular ORB-SLAM's tracking accuracy has been compared to ORB-SLAM with ground-truth depth-maps and to ORB-SLAM with estimated depth-maps. This comparison shows that adding estimated depth-maps increases the tracking accuracy of ORB-SLAM, but not as much as the ground-truth depth images.

The same setup is tested on WEpod images. The CNN is fine-tuned on 7481 Kitti images as well as on 642 WEpod images. The performance on WEpod images of both CNN versions are compared, and used in combination with ORB-SLAM. The CNN fine-tuned on the WEpod images does not perform well, missing details in the estimated depth-maps. However, this is enough to solve the initialization problem of ORB-SLAM. The combination of ORB-SLAM and the Kitti fine-tuned CNN has a better tracking accuracy than ORB-SLAM with the WEpod fine-tuned CNN. It has been shown that the initialization problem on WEpod images is solved as well as the tracking accuracy is increased.

These results show that the initialization problem of monocular ORB-SLAM on WEpod images is solved by adding the CNN. This makes it applicable to improve the current localization methods on the WEpod. Using only this setup for localization on the WEpod is not possible yet, more research is necessary. Adding this setup to the current localization methods of the WEpod could increase the localization of the WEpod. This would make it safer for the WEpod to navigate through traffic. This research sets the next step into creating a fully autonomous vehicle which reduces traffic accidents and increases the mobility of people.

Contents

1 Introduction

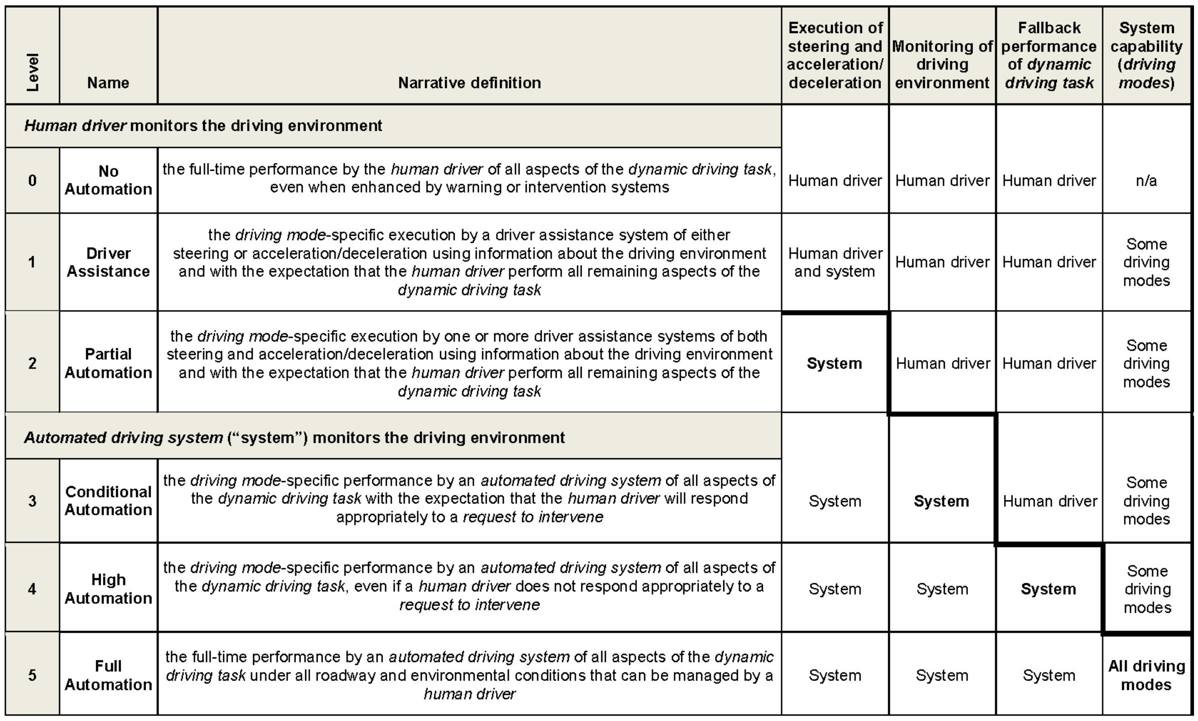
The amount of deaths worldwide due to road traffic accidents has risen over the course of 20 years (1990-2010) by 46 % to 1.3 million yearly deaths [36]. This is the price that we to enjoy the advantages of driving. Questions arise about the causes of these traffic accidents and if it is possible to lower this amount of deaths while still keeping the advantages of transportation. Research has been done into the causes of these traffic accidents. They concluded that the main causes of road traffic accidents are the drivers fault. According to a recent report [53] 65 %, and even possibly up to 92.6 %, of fatal traffic accidents can be attributed to human error.

A lot of effort has been put into reducing the number of fatal traffic accidents. Several methods to achieve this have been found. One method of reducing the amount of deaths is by adding safety features to the car. These reduce the chance of fatal injuries when a traffic accident has happened. Examples of these safety features are for instance the addition of seat-belts, airbags or improving the shape and material of the frame. This solutions do not not tackle the actual problem, but more reduces the severity of the consequences.

Another method of reducing the amount of deaths is by tackling the traffic accidents itself. Removing the cause of these human errors would have a great impact on the amount of traffic accidents. Most of the traffic accidents can be attribute to human error. Examples of these causes are inattention and fatigue. The human errors can divided into four groups [49] : slips (misread road signs), lapses (fail to recall road just travelled), mistakes (underestimate speed of oncoming vehicle) and violations (exceeding the speed limit).

A possible solution to these human failures is to support the human in the driving task. There are a lot of advanced driver-assistance systems (ADAS) coming into the vehicle market. Examples are adaptive cruise control, anti-lock braking systems, emergency braking systems and electronic stability control. Instead of supporting the human while driving, replacing the human as a driver with a driving system is also a solution to the problem. This driving system would for instance never get tired and is always attentive. Replacing the human driver would also result in a lot of other advantages. For instance, the human that usually drives the vehicle can now use its time in traffic to its advantage. The time inside the vehicle, usually wasted on driving, can now be used to relax, study or work. Studies have shown that Americans spend an average of 50.6 minutes per day driving [52] and Europeans even more then a hour per day [41]. Also does it give people without the ability to drive the vehicle, for instance elders or children, the chance to safely use the vehicle to get to their destination without being dependant on others.

In the car industry, there are a lot of companies implementing ADAS in their vehicles. They try to assist the driver more and more by improving ADAS. These vehicles driving systems share the control with the human, taking over more and more tasks over from the human. They climb their way through levels of automation in driving defined by the SAE. Level 0 is total human control and level 5 is total control by the systems [26]. The explanation of the five levels can be found in figure 1.1. Another strategy that companies implement is directly removing the human control. They immediately try to create a vehicle that fits in the forth level of automation. Attempts of these level 4 automated vehicles are the Heathrow shuttles and Rotterdam Rivium shuttles, which only drive on a dedicated lane without other traffic, or the Navya autonomous shuttle, capable of driving in between traffic in certain situations.



Level	Name	Narrative definition	Execution of steering and acceleration/ deceleration	Monitoring of driving environment	Fallback performance of *dynamic driving task*	System capability (*driving modes*)
Human driver **monitors the driving environment**						
0	No Automation	the full-time performance by the *human driver* of all aspects of the *dynamic driving task*, even when enhanced by warning or intervention systems	Human driver	Human driver	Human driver	n/a
1	Driver Assistance	the *driving mode*-specific execution by a driver assistance system of either steering or acceleration/deceleration using information about the driving environment and with the expectation that the *human driver* perform all remaining aspects of the *dynamic driving task*	Human driver and system	Human driver	Human driver	Some driving modes
2	Partial Automation	the *driving mode*-specific execution by one or more driver assistance systems of both steering and acceleration/deceleration using information about the driving environment and with the expectation that the *human driver* perform all remaining aspects of the *dynamic driving task*	**System**	Human driver	Human driver	Some driving modes
Automated driving system **("system") monitors the driving environment**						
3	Conditional Automation	the *driving mode*-specific performance by an *automated driving system* of all aspects of the *dynamic driving task* with the expectation that the *human driver* will respond appropriately to a *request to intervene*	System	**System**	Human driver	Some driving modes
4	High Automation	the *driving mode*-specific performance by an *automated driving system* of all aspects of the *dynamic driving task*, even if a *human driver* does not respond appropriately to a *request to intervene*	System	System	**System**	Some driving modes
5	Full Automation	the full-time performance by an *automated driving system* of all aspects of the *dynamic driving task* under all roadway and environmental conditions that can be managed by a *human driver*	System	System	System	**All driving modes**

Figure 1.1: The SAE five stages of automation in driving with explanation [26].

Another company that works on fully automated vehicles is Robot Engineering Systems (RES). RES has developed the WEpod in collaboration with the TU Delft and other partners. The WEpod is a shuttle for 6 passengers capable of driving on its own and is depicted on the front page. The WEpod shuttle drives people between the campus in Wageningen and the train station Ede-Wageningen. It is capable of driving in between traffic with a speed of 25 km/h. The WEpod is used for a lot of research on new and improved methods of driving autonomously and comfortably. To be able to improve its autonomous driving capabilities, research on different automated driving fields is done in collaboration with the TU Delft. Examples of these automated driving research fields are path-planning, object-detection, decision-making and localization.

A WEpod shuttle uses a setup of different sensors to be able to sense its surroundings. It has 9 radars divided around the vehicle. These radars sense in one plane the distance to objects and they estimate the speed of the objects they sense. The WEpod has 6 lidars placed around the vehicle. Each lidar senses 3D point distributed on four horizontal planes. Next to the radars and the lidars, also 9 cameras are distributed around the outside of the vehicle. Internally is the vehicle equipped with a GPS and IMU unit and enough CPU/GPU setups to provide enough computing power to process all the data.

All these sensors and computers on the WEpod are used to safely and autonomously drive in mixed traffic on public roads. However, there are still challenges for the WEpod. One example is diverting from its predetermined path to drive around an obstacle. Currently it would detect the obstacle and stop. It would wait until the obstacle moves so it can resume its path. It cannot overtake others by diverting its path around the object in front of it. One of the reasons that it cannot do this is because of its poor localization performance.

The WEpod currently localizes itself inside a pre-built map of the area. This is implemented by a third party named Ibeo. They do this by using the points obtained with the lidars and the GPS information. However, this proves to have multiple downsides. Not only is it costly to pay Ibeo for each new map, the system also does not preform robust enough using these sensors.

One of the reasons why the system of Ibeo does not preform robust enough is because localizing using lidar sensor information proves to be inaccurate in certain scenarios. An example of such a situation is when driving in a street with similarly shaped buildings on both sides. Distinguishing one building from the next based on its shape in the lidar pointcloud is then very hard. Determining you location in such a street accurately is not possible. Other downsides of using a map based on four plane lidar points is that is produces a sparse distribution of points on the map and that the location of the road is not described on the map.

Another reason why the system of Ibeo does not preform robust enough is because of the GPS. The GPS suffers from different types of obstacles that block the GPS signals from the satellites. One type of objects that block the GPS signals are leaves. This is caused by the water in the leaves blocking the GPS signals. Another example of objects that block the GPS signal are large buildings. When driving through cities with a high density of large buildings, no direct line of sight is available between the GPS satellites and the GPS unit. The GPS unit only gets the signals reflected of the buildings, resulting in poor localization by the GPS. Dalip [4] has mapped out the different effects of environmental parameters on GPS signals. He proved that localization by GPS suffers from these signal blocking objects.

So the localization method of the WEpod needs to be improved to allow safe and flexible navigation through mixed traffic. RES wants to be able to create there own maps of the surrounding to make them independent of Ibeo. The process of creating a map of the surrounding and localizing yourself on this map is called Simultaneous Localization And Mapping (SLAM). RES wants to use a SLAM method that takes sensor data from the WEpod, creates a map of the surrounding environment and localizes itself on this map.

There has been a lot of research done into different SLAM methods. Preforming SLAM using different types of sensors is an example. One of the sensors used to preform SLAM with is the radar. [44] is an example of performing SLAM using radar sensors. Performing SLAM based on lidar sensors, comparable to the method the WEpod localizes now, is also possible. The lidars can have different amount of planes in which they scan with different densities. This results in different performances of lidar based SLAM systems. Finally another sensor to do SLAM with is a camera. These vision-based methods use the image stream of a camera to estimate a map of the surrounding and the path travelled by the camera. Vision-based SLAM methods that only use the image-stream of one camera as an input are called monocular vision-based SLAM methods.

RES has chosen to investigate monocular vision-based SLAM. Localizing and building a map using the lidars and GPS proves to not be adequate. It is assumed that preforming SLAM based on the radars would not preform better than the lidar based localization and mapping. This because the radar returns similar sensor data as the lidars, but than in less planes (only 1) and less dense. Vision-based SLAM however provides a very different type of data then the lidar, much denser and not bound to certain planes. Vision-based SLAM methods have been proven to work on datasets consisting of camera's on a car driving on the road [39]. A camera is also much cheaper than a lidar setup, resulting in cost benefits. So it would be ideal to be able to preform SLAM using camera data by vision-based SLAM. There is no depth calculating setup available on the WEpod, so the localization investigation of RES is focused on monocular vision-based SLAM.

Monocular vision based SLAM algorithms can be divided into two parts [59], the tracking part and the mapping part. The tracking part estimates the pose of the new image and the mapping part creates and constantly optimizes the surrounding map. The existing monocular vision-based SLAM methods can be divided based on their working. They can be direct or feature-based and sparse or dense. Direct methods do the tracking part by minimizing a photometric error directly calculated from the intensity differences between images. Feature-based methods first extract features from the images and do the tracking part by minimizing a reprojection error of these feature points. They can also be classified as sparse or dense methods, based on the density of points used to create the map. Some methods select certain points from the image and use only these to create the map (sparse methods), while others try to use as many points as possible to create the map, creating a much denser map (dense methods).

During a literature study done in cooperation with RES, four state of the art monocular vision-based SLAM algorithms have been investigated to choose one for this thesis [59]. The state of the art monocular vision-based SLAM algorithms that where investigated are: Large Scale Direct SLAM (LSD-SLAM) [9], ORB-SLAM [39], Direct Sparse Odometry (DSO) [10] and Semi-direct Visual Odometry (SVO) [17]. From the conclusion of the literature study it can be taken that ORB-SLAM is the most suitable of the four as it preforms better than the rest. The implementation of monocular ORB-SLAM is also openly distributed within ORB-SLAM2 [40], together with the source code. So ORB-SLAM is investigated as an option for monocular vision-based SLAM algorithm on data from the WEpod.

ORB-SLAM on data from the WEpod has been tried out during an internship at RES [58],. Unfortunately this internship proved that the performance of ORB-SLAM on WEpod data was nowhere near the performance of monocular ORB-SLAM on data from the Kitti dataset [20]. The Kitti dataset consists among others of images from a camera on a driving car. Initializing a starting scene to preform the tracking and mapping from is to difficult. Even though ORB-SLAM has a more robust initializing method than the other state of the art monocular vision-based SLAM methods [59]. ORB-SLAM tries to initialize using either the fundamental matrix, which works well in non-planar scenes with high parallax, or using the homography matrix, which works well in planar scenes with low parallax. Both initialization methods do not pass the test ORB-SLAM uses to check if the initialization is correct, resulting in ORB-SLAM trying to initialize over and over again.

This initialization problem is a problem that not only occurs on the WEpod data. The authors of many SLAM algorithms have tried to tackle this problem. A set of 2D points that describe a 3D scene are used for initialization. Using 3D points for this initialization would be easier. These 3D points can be obtained by adding depth information as an input to ORB-SLAM [59]. Adding depth information makes the initialization of the starting scene a lot easier. During the internship at RES [58], it has been proven that indeed adding depth information to the inputs of ORB-SLAM solves the initialization problem.

The depth information required can be supplied using different methods. One common way is adding a depth-map to the input. This depth-map can be obtained by using for instance infrared sensors (like with the Kinect) or by linearly interpolating between the lidar points from a 64-plane lidar. Another method of obtaining the depth information is by supplying stereo images. By finding stereo matches of points in the stereo images, depth can be estimated and thus is the initialization problem much easier.

Unfortunately, the WEpod does not have a stereo camera setup on-board, and neither does it have other sensors which makes it possible to achieve a dense depth map of the images. It does have lidars and radars estimating depth, but these are much too sparse to obtain a dense enough depth-map for ORB-SLAM. It would also be to costly to adapt the sensory setup of the vehicle. Therefore, the depth information must come from monocular images.

Obtaining depth from monocular images is not an easy task. Obtaining depth from one image stream can be done with structure-from-motion algorithms. They treat the different images trough time as ‘stereo’ pairs, finding stereo matches between them. Using these stereo matches, they try to estimate the pose difference between the two images, inertly estimating the depth of the stereo matched points. This however is very similar to how ORB-SLAM algorithms treat the input image stream. They also try to estimate the depth of points in the image using such a method, and this proves to not be accurate enough. So another method of estimating the depth in monocular images is needed.

Hoiem [25], and Liu [33], have tried to estimate the depth in an image based on manually defined features and probabilistic graphical models. These methods are heavily based on assumptions about the surrounding environment in the images, making them far from robust and not applicable for estimation of a depth-map of a road scene.

Using convolutional neural networks (CNN’s) to estimate the depth in an image is also a possibility, that outpreforms the previous methods. The rise in computational power of the last years has boosted the capabilities and research in neural networks. A lot of research has been done in solving automated driving problems with neural networks, such as object-detection, path-planning and even depth-map estimation of images. Using a CNN would make it even possible to achieve a cooperation between these CNN’s doing different things in the vehicle, they could possibly easily share information with each other. A CNN that estimates depth in a single image has been proven to be possible, as multiple people have created networks doing this. Examples of these depth-estimating CNN’s can be found in chapter 2.

To be able to use depth estimating CNN’s, groundtruth depth-maps, stereo images or accurate pose information is necessary. The pose information from the WEpod is not accurate enough to use with a depth estimating CNN setup, if it was this whole research would not be necessary. There is also no stereo image setup available on the WEpod. There is a separate stereo sensor available, the Multisense S21. However this cannot easily be used to fine-tune depth estimating CNN as the CNN would than be fine-tuned on data from a different camera than it would predict on. This would not necessary mean that it would preform good as the camera’s would not only look from different perspectives but they also have different intrinsic parameters, field of views and could react different to different lighting circumstances.

The stereo camera setup however can be used to obtain groundtruth depth-maps of the WEpod images. This can be done by placing the stereo camera setup above the WEpod camera facing the same direction such that the view of both cameras overlap as much as possible. The transition between both the camera frames (the left camera of the stereo camera and the WEpod camera) can be estimated using stereo calibration. Then enough information is present to project the depth points seen by the stereo camera setup to the WEpod camera image, creating a depth-map of the WEpod images. See for more explanation chapter 3.

Another possibility of fine-tuning the depth estimating CNN of CNN-SLAM for prediction of WEpod images is by using a different but similar dataset. Such a dataset is the Kitti dataset [20]. It consists of images from a camera fixed on a car and lidar points from a 64-plane lidar. The car drives around in different types of road scenes, from dense cities to open highways in Germany. The dense lidar point cloud can be used to create a depth-map of the images, which then can be used for fine-tuning the depth estimating CNN. This depth estimating CNN fine-tuned on Kitti data can be used to predict the depth in WEpod images.

This means that the depth estimating CNN can be used to estimate the depth in the monocular images. The CNN setup of CNN-SLAM is one of the largest depth estimating CNN's but also the best performing. Therefore the CNN of CNN-SLAM will be used to estimate the depth in the images and this depth can then be passed into ORB-SLAM. This setup is depicted in figure 1.2, showing that this setup is still monocular, even though ORB-SLAM is not running in a monocular mode.

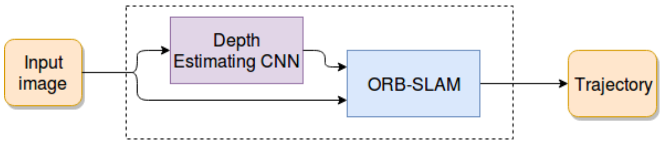


Figure 1.2: Diagram showing how the CNN can be added with a monocular vision-based SLAM algorithm (ORB-SLAM2 here as example), while keeping the whole system monocular. The images from the camera stream go into the CNN to estimate its depth-map. This depth-map is then together with the image forwarded as inputs for ORB-SLAM2.

To summarize, the current localization method of the WEpod is not robust enough to allow safe and flexible navigation through mixed traffic. To improve the localization of the WEpod, the monocular vision-based SLAM algorithm ORB-SLAM [39] has been investigated. Unfortunately, ORB-SLAM fails to initialize the starting scene when run with WEpod images. It thus fails to perform SLAM on WEpod data. It has been shown that the problem of not initializing the starting scene can be solved by adding depth-maps of the images to the input of ORB-SLAM [58], [59]. The WEpod however does not have any sensors capable of creating an accurate and dense enough depth-map of its images.

Therefore, the goal of this thesis is to create and add a depth estimating CNN to ORB-SLAM. This addition of a depth estimating CNN to a monocular vision-based SLAM algorithm should improve the tracking accuracy when using input images from the WEpod. The depth estimating CNN needs to be fine-tuned such that it can estimate the depth in WEpod images.

The depth estimating CNN of CNN-SLAM [31] is suited to tackle this challenge. The authors have trained and tested its performance on the NYU dataset, which consist of indoor scenarios. The performance of the CNN needs to be tested on a dataset of road scenarios. The Kitti dataset is used, as it includes 64-plane lidar data that makes it easy to create accurate depth-maps. The CNN is fine-tuned on the Kitti dataset and then predicts on Kitti images.

The depth-maps for the WEpod images can be created using a stereo setup. These depth-maps can be used to fine-tune the CNN. The performance of the CNN fine-tuned and predicting on WEpod images can be compared with the performance of the CNN fine-tuned and predicting on Kitti images.

It is also interesting how the CNN performs when it is fine-tuned on Kitti images and predicts WEpod images, cross-domain. This could make the training of the CNN much less of a hassle, making fine-tuning for each similar dataset unnecessary. One trained CNN could possibly be used for many more applications without needing retraining, for instance one CNN can be used for multiple vehicles. Therefore the performance of the CNN fine-tuned on Kitti images and predicting WEpod images is compared to the performance of the two previous setups.

The addition of the depth estimating CNN on ORB-SLAM is first investigated on the Kitti dataset. Monocular ORB-SLAM does initialize on images from the Kitti dataset. The effect of adding depth information, ground-truth and estimated with the CNN can be investigated This can be done by comparing the tracking accuracy of ORB-SLAM only on Kitti images, with ORB-SLAM on Kitti images together with depth-maps from the lidar points and together with depth-maps estimated using the CNN.

Then the addition of the depth estimating CNN on ORB-SLAM with WEpod images can be investigated. The tracking accuracy of ORB-SLAM only on WEpod images can be compared to ORB-SLAM on WEpod images and the corresponding depth-maps estimated with the CNN.

The scientific work related to this thesis can be found in chapter 2. Chapter 3 describes the methods and setups used. The experiments and their results are presented in chapter 4. These results are interpreted and discussed in chapter 5. Chapter 5 also concludes if this research has been successful in solving the initialization problem of ORB-SLAM with a depth-estimating CNN.

2 Related Work

2.1. Introduction

As described in the introduction, the localization capabilities of the WEpod need to be improved. This can be done with monocular SLAM methods. However, even the state of the art methods have trouble initializing on WEpod images. In order to perform SLAM with the camera's from a WEpod, a monocular vision-based SLAM method need to be improved. Especially the initialization needs to become much more robust. As found during my internship [58], my literature study [59] and by many state of the art SLAM methods [8] [55] [17] [40], this can be done by adding depth information to the inputs. A monocular vision-based SLAM method needs to be chosen to work with. The are multiple methods in existence. For this research, a well-performing, state of the art and openly available monocular vision-based SLAM method is needed. In section 2.2 the working of state of the art monocular vision-based SLAM methods is explored so one can be chosen in the end of this chapter.

A method to obtain the depth information needs to be found. This is a considerable challenge as the WEpod does not have any sensors that can accurately and densely estimate the depth in its images. Using structure-from-motion algorithms is a conventional way of obtaining the depth-map from an image stream. How these algorithms try to estimate depth from images is comparable with how SLAM methods estimate the pose. Therefore, depth-maps estimated by structure from motion algorithms will probably not improve the SLAM methods. It will encounter the same challenges. Other depth estimating methods need to be explored.

Another way of depth estimation is for instance done by [25] and [33]. They define probabilistic graphical models and try to manually define features in the images. Using these features and models, they try to estimate the depth in images. This proves to be very limited, as these methods are heavily dependent on assumptions about the surrounding environment in the images. This makes them far from robust and not applicable for depth estimation in the WEpod.

In this thesis a convolutional neural network (CNN) is used to estimating depth in images. Due to the rise in computational power over the last couple of years, research and uses for convolutional neural networks has exploded. CNN's are used for many different types of tasks within autonomous vehicles. Examples of these tasks are object-detection, path-planning and end-to-end driving with one network [2]. This end-to-end driving is done by letting the network take camera images as inputs which it maps to inputs for the motors, steering etc. As CNN's are already used in autonomous vehicles, using a CNN for depth estimation of images would fit nicely with the other CNN's. This would allow cooperation between CNN's in the feature. Examples of depth estimating CNN's are investigated in section 2.3. This makes it possible to choose a depth-estimating CNN to work with in this thesis in the end of this chapter.

The addition of a depth estimating CNN to a SLAM network is not completely new. It has been done before in the CNN-SLAM [51] algorithm, which unfortunately has not been openly shared. The depth estimating component of CNN-SLAM [31] is investigated as an optional CNN to use in this thesis in section 2.3. CNN-SLAM shows that adding a depth estimating CNN to monocular SLAM does indeed improve the mapping capabilities of SLAM for indoor scenarios. They only use the depth estimating network to estimate the depth in the keyframes, which are only used for the creating the map. The CNN is not used on all frames to improve the tracking accuracy. Furthermore, it has not been proven that the same concept works with outdoor road scenery images. In this thesis a depth estimating CNN will be used on all the frames to improve the initialization and tracking performance.

The concept of this thesis will also be investigated on another dataset than the WEpod images. This is done because there are no groundtruth depth-maps of the WEpod images suitable to perform SLAM with available. Also does monocular SLAM without CNN not initialize on the WEpod images. When SLAM together with the depth-estimating CNN does initialize on WEpod images, its tracking accuracy cannot be compared to see if it has improved. The fact that it would initialize is an improvement itself, but a quantitative measure would be nice. Therefore, the KITTI dataset [20] is used to compare the tracking accuracy of monocular SLAM without depth-estimating CNN to monocular SLAM with depth-estimating CNN and to SLAM with images and groundtruth depth-maps.

The KITTI dataset has an odometry dataset, which is used in this thesis. It consists of 22 sequences, combined covering 39.2 km of different types of roads in Germany in a frequency of 10 fps. Each sequence contains not only color and gray scale stereo images, but also Velodyne 64-plane lidar scanner data, and GPS/IMU localization information. The exact placement of the sensor can be found on their site [19]. The ground truth pose information is obtained using the GPS/IMU data, which is projected into the coordinate system of the left camera. The 64-plane lidar points can be used to obtain ground-truth depth-maps of the images, see section 3.3.

2.2. Vision-based SLAM

SLAM stands for Simultaneous Localization and Mapping. Vision-based SLAM algorithms aim to estimate the six degree pose (translation and rotation) of a camera and create a map of the surrounding environment, using images. SLAM has been an active research field for a few decades already [54]. There was a big boost lately due to the increase in computation power and its application in new fields such as robotics, drones and automated driving. Vision-based SLAM is a very cheap way of doing SLAM in comparison to other options such as using lidar data. A 64 plane lidar is a factor 100 more expensive than a camera.

A lot of people have investigated the SLAM problem. This resulted in the creation of different SLAM algorithms using different types of inputs. A lot of these different SLAM methods can be found on different websites created for the cataloging and comparing of the different SLAM methods. Examples of such sites are the KITTI dataset site [19] (provides performance comparisons) and the OpenSLAM site [3] (listing SLAM algorithms from before 2015). Unfortunately, not all SLAM algorithms are openly available for research purposes as some are used for consumer products. Monocular vision-based SLAM are especially interesting because of the sensory limitations of the WEpod.

Four state of the art monocular vision-based SLAM methods are LSD-SLAM [9], ORB-SLAM [39], Semi-direct Visual Odometry (SVO) [17] and Direct Sparse Odometry (DSO) [10]. These four methods resemble different ways monocular vision-based SLAM algorithms work. These four methods are able to make large scale maps of urban locations instead of only small indoor scenes. There are however many other SLAM methods, for example some other high scoring monocular SLAM methods from the KITTI site [19] are: FVO [42], PMO [13] and FTMVO [38].

All well performing real-time SLAM methods [9], [39], [17], [10], [40] use keyframes. Instead of using all the frames from the image stream to create the map, they try to filter out a select few frames that contain as much different information as possible. Each SLAM algorithm has their own way of determining when to select a frame as a keyframe. This is usually based on the distance travelled from the last keyframe or the amount of new information in the frame. The keyframes usually consist of the image, the inverse of the estimated depth, the variance of the estimated depth and the pose of the keyframe. The redundant frames are usually used to update the points on the last keyframe. Using keyframes allows real-time performance of the SLAM algorithms. The algorithms can keep up with the image stream from the camera. Another tactic to achieve real-time performance is multi-threading the pose estimation part and the mapping part.

Monocular vision-based SLAM methods can be divided into two types based on how they extract the necessary information from an image. They can either use the whole image, called direct methods, or by first extracting certain features from the image, called feature-based methods. The difference between direct and feature-based methods is clearly depicted in figure 2.1, adopted from the paper of LSD-SLAM [9].

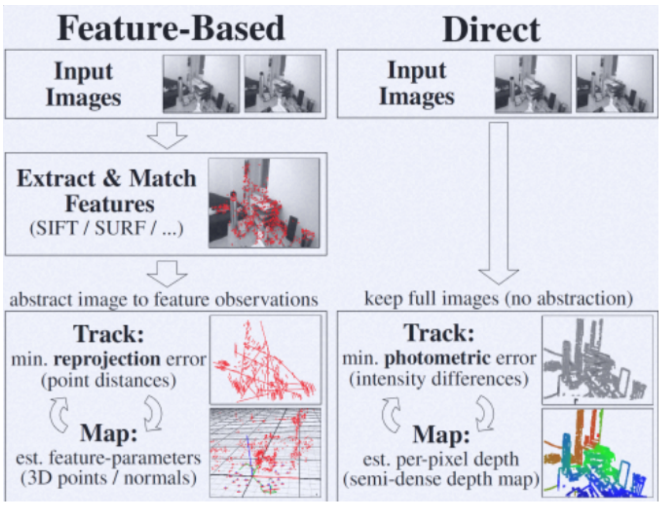

Feature-Based
Direct
Input Images
Input Images
Extract & Match Features (SIFT / SURF / ...)
abstract image to feature observations
keep full images (no abstraction)
Track: min. reprojection error (point distances)
Map: est. feature-parameters (3D points / normals)
Track: min. photometric error (intensity differences)
Map: est. per-pixel depth (semi-dense depth map)

Figure 2.1: Example of the difference between feature-based and direct SLAM. Obtained from the LSD-SLAM paper [9].

Feature based methods create an intermediate representation of the image by extracting a set of keypoints from the images or by searching line segments. This intermediate representation is fed into a geometric model based on a geometric error, they are more robust against geometrical noise. These types of SLAM methods use optimization methods like bundle adjustment [12]. ORB-SLAM is an example of a feature based SLAM method. It extracts ORB points from the images for the fast extraction and view orientation invariance.

Direct methods do not use an intermediate representation of the image, but directly use the image in a photometric model. This is optimized by Gauss-Newton optimization methods. A photometric model, based on photometric calibration as in [11], consists of pixel-wise attenuation factors V describing the vignette (the effect where the light brightness changes from the center to the outside of the image), the exposure time t, and a camera response function G which maps the irradiance image B (the amount of energy on each pixel) to the intensity of the pixel I: $I(x) = G(tV(x)B(x))$ [11]. G and V can be found by performing photometric calibration with the camera. The details of the calibration can be found in the paper [11]. LSD-SLAM [9] and DSO [10] are examples of direct methods. They use the photometric model to account for lens attenuation, gamma correction and exposure times and thus being more robust against noise for these parts.

SVO [17] is a special case, they call themselves semi-direct. They use a photometric model and use the intensity all over the image. However instead of using all this information, they choose a sparse set of points distributed over the highest intensity gradients (edges and corners) all over the image. They then refine these points using bundle adjustment, trying to get the best sides of both feature based and direct methods.

Another way of categorizing the monocular SLAM methods is by looking at the density of the map created. Dense SLAM methods try to create a dense as possible map. LSD-SLAM [9] is an example of a dense monocular SLAM method. The methods that create sparser maps are categorized as sparse SLAM methods. They use a sparse independent set of points for the creation of the map where usually the relative position of points is disregarded. Examples of sparse SLAM algorithms are ORB-SLAM [39] and DSO [10]. SVO [17] is again an odd one out, calling themselves semi-dense. They use a subset of the pixels in an image together with their connectivity information (the fact that they are edges or corners) to create the map.

The inner working of monocular SLAM algorithms can be split up into four parts:

- The initialization part kicks off the algorithms by initializing the map and the starting pose.
- The tracking part kicks in which estimates the pose of each frame and sends the keyframes to the mapping part.
- The mapping part creates the map using the keyframes
- The loop-closure part tries to detect loop closures and relocalizes when tracking failed.

Each part is examined slightly more in the following paragraphs.

The initialization part of the SLAM algorithms is done with the first few frames. An initial pose and map needs to be found to start SLAM, which then starts improving and elaborating the poses and map. During the initialization, only 2D matched points in two images are known. The pose transformation between two images and the depth of the points in the images are unknown. If two of these three things are known, the last one can be calculated. For instance: if the 2D points in images are correctly matched and the pose between the images is known, then the depth can easily be calculated. The initialization is such a challenge because two of the three thing are unknown and need to be estimated. Later on in the SLAM process the algorithms has a map full of points with a known 3D position. These map points can be found in the new images and used to estimate the pose and the depth of new points. However, no known map points are available during the initialization. A common, basic way of initializing is by starting off with a keyframe with a random depth and a large variance. LSD-SLAM [9] is one of the algorithms that starts this way, claiming that after a few keyframes with enough translational camera movement, it will converge to the right values. However, [10], [17] and [39] all report problems with this way of initialization of LSD-SLAM. Therefore, ORB-SLAM [39] initializes differently by implementing two ways of initialization and choosing the best one.

The tracking part of the SLAM algorithms estimates the pose from the new frame to the previous frame [17] or keyframe [9]. This can be done for instance by minimizing a variance-normalized photometric error [9] [10], by using image to model alignment (minimizing the intensity differences of pixels of the points) [17] or by minimizing the geometric error of matched feature points in images [39]. The tracking part also decides if the frame is used as a keyframe. It estimates depth in keyframes from the pose calculations .

The mapping part of the SLAM algorithms takes the keyframes and creates the map. This map is created by optimizing the poses of the keyframes and the points in the keyframes. The image points in the keyframes, together with the poses of the keyframes make it possible to project the image points to the 3D space to create the map. The keyframes are first optimized using spatial regularization and outlier removal [7]. Then they are added to a graph that represents the map. The poses and points of the keyframes in this graph are then optimized using Pose Graph Optimization [29] or bundle adjustment [12]. A common tactic [39] [10] is to create a lot of keyframes to be able to do robust and accurate tracking, and then to marginalize these keyframes to keep the computational cost of the optimization's low. An important note to make when looking at monocular SLAM is that as they do not get any depth information, the scale of the map is a free parameter. The absolute scale of the map cannot be estimated and thus tends to drift.

The loop-closure and relocalization part of the SLAM algorithms constantly tries to detect loop-closures and tries to relocalize when tracking is lost. By trying to detect loop-closures, monocular vision-based SLAM algorithms try to solve the problem of scale drift. They compare the current frames to the keyframes from the map, trying to find frame matches. When a match is found between the frame and a keyframe, the scale and poses of the current frame and the keyframes inside the loop can be adjusted. Relocalization after loosing track of the poses is usually done in the same way.

Improvements on the vision-based SLAM algorithms have been made by stepping away from the monocular limitation and using stereo images [8] [55] [17] [40] or depth-maps [40] as inputs. With these extra inputs, they can extract depth information to obtain the absolute scale of the map. The depth information is also used with the initialization, which is no longer a complicated issue with the depth information. The performance of the versions with the depth information improved compared with the monocular versions.

2.3. Depth estimating CNN's

A convolutional neural network (CNN) is a type of neural network suited for working on images. Images can be of hundreds by hundreds of pixels. This means that they live in a feature space of hundred thousands of features. Passing images to neural networks consisting of fully connected layers would mean that the network would need billions of weights to actually do something useful. The convolutional part of CNN's makes them useful when the data is not location specific in the image, the CNN's are location invariant in image space.

A CNN can consist of a combination of convolutional layers, ReLU layers, pooling layers and other layers such as fully connected layers. Especially the convolutional layers result in needing a lot less of parameters by taking advantages of certain characteristics of images. The filters of these convolutional layers can for instance recognize a certain shape in the image by convolution.

A great explanation of Convolutional Neural Networks is given in the course notes of the Stanford course 'CS231n Convolutional Neural Networks for Visual Recognition' of Fei Fei Li, Justin Johnson and Serana Young [15]. Methods for creating an intuition of how a certain CNN works can be found in [21], [50], [56], [47], [48], and [37]. There has been proven that CNN's are the go to type of neural network when working on images [45]. Other interesting research into the working of CNN's themselves are attempts to fool them [22], show that they learn correspondences in between images [35] and show that CNN's can outperform humans in classification tasks [27].

An interesting depth estimating CNN is the one used in CNN-SLAM [51]. The depth estimating CNN itself is explained in [31]. CNN-SLAM uses its CNN to estimate the depth-maps of the keyframes in the SLAM system. They use this depth-map together with the keyframes to perform more robust and accurate SLAM. The result of the SLAM system is combined with the result of a separate semantic segmentation estimating CNN to create semantic segmented maps. The CNN is based on the ResNet-50 CNN [24], with a self designed decoder. This is a very large but well performing network compared to other depth estimating networks. The whole setup has been used for indoor scenes, but not for outdoor road scenes. On the indoor scenes, they report better performance than monocular LSD-SLAM [9] and ORB-SLAM [39], but they did not report if and how they have dealt with the independent scaling variable in the monocular SLAM maps.

Other interesting depth estimating CNN's are the ones that learn unsupervised. This can be done by training on stereo images [18]. Only the left image is used for the forward pass, after which points of the left image are projected to the right image to obtain a loss function based on the reprojection error.

Another way of training a depth estimating CNN in an unsupervised way is by combining it during training with a pose estimating CNN [57]. The pose estimating CNN is used to estimate the pose difference of the next and previous frame in the camera sequence. The current frame is passed through the depth estimating CNN to obtain the estimated depth. Using the estimated depth and the pose difference, points from the current frame can be projected to the next and previous frames. From these projections, a photometric error is calculated and used as the error to be minimized in the loss function.

An interesting idea is presented in [6] to split the depth-estimating CNN into two CNN's. The first CNN takes the image as an input and estimates the depth on a coarse level. The second CNN takes the image and the coarse depth as inputs and uses these to estimate the depth on a much finer level. The idea behind this is that the coarse CNN filters are smaller but the layers are deeper, while the filters of the fine CNN are slightly larger to incorporate more spatial information but the layers are not as deep as it already knows the rough depth. Unfortunately does this method not improve the result, they report comparable, and even slightly worse, results compared to the depth estimating CNN's above. In their results, it can be seen that the addition of the fine CNN only improves the result in the metrics slightly while the difference in the depth-maps visually is huge. The fine CNN creates much sharper edges and corners, but also apparently introduces additional errors, resulting in only a very slight improvement in the metrics.

2.4. Conclusion

A monocular vision-based SLAM method for this thesis needs to be chosen. The state of the art monocular vision-based SLAM method ORB-SLAM [39] has been chosen. ORB-SLAM has proven to be one of the best performing state of the art monocular SLAM methods [59], has the code openly available for research purposes and has an improved method of initializing. Its implementation in the openly available ORB-SLAM2 code [40] is used in this thesis, as in ORB-SLAM2 it is also possible to add depth-maps of the images to the inputs.

As for the depth-estimating CNN used for this thesis, the CNN of CNN-SLAM [31] has been chosen. It is the best performing depth estimating CNN of the ones described here [59]. Even though it does not use skip-connections and others do, it still performs better. It has also been shown that this network works well in cooperation with SLAM on indoor scenarios. This network is substantial larger than the other depth estimating networks. It still fits on the GPU used in this thesis, so this is not a problem. It's setup is openly available together with the the parameters it gained when training on the indoor NYU dataset.

By combining the depth estimating CNN of CNN-SLAM with the monocular version of ORB-SLAM2, an attempt has been made to improve the tracking accuracy of the monocular version of ORB-SLAM2. Monocular vision-based SLAM will be improved with a depth estimating CNN. The whole setup will be kept monocular. It will improve the tracking accuracy on outdoor road scenery images. This has not been done before.

3 Methods

The current localization method of the WEpod needs to be improved. This can be done by using the monocular version of ORB-SLAM, an algorithm that creates a map and localizes itself on this map using one camera. The working of ORB-SLAM is explained in depth in section 3.1. Monocular ORB-SLAM has been proven to work using road scenery images from the Kitti dataset, by the authors of ORB-SLAM [40] and in our experiments.

However, when monocular ORB-SLAM is applied to images from the WEpod camera, the algorithm does not work as well. This has been proven during my internship with the WEpod [58] and can be seen in our experiments. The algorithm fails to initialize the surrounding scene and constantly keeps on trying to reinitialize. Something about the WEpod images is different compared to Kitti images, causing this initialization problem. The exact reason why the initialization fails, is not the scope of this thesis. The goal of this thesis is to find a solution to the initialization problem such that ORB-SLAM can be used to improve the localization of the WEpod.

The methods used to achieve this goal are explained in this chapter. First the working of ORB-SLAM itself is investigated in section 3.1. Then the working of the depth-estimating CNN is explained in section 3.2. Ground-truth depth-maps are required for the fine-tuning of the CNN. The way these are obtained is explained in section 3.3. In section 3.4, the method of determining the hyper-parameters for fine-tuning can be found. Finally the overall approach of this thesis is elaborated in section 3.5.

3.1. ORB-SLAM

ORB-SLAM [39] is a sparse, feature-based SLAM method that utilizes a bundle adjustment method. It is able to perform monocular vision-based SLAM in real-time. As the name suggests, it is based on ORB [43] features. These provide fast extraction and matching while being view orientation invariant. ORB-SLAM consists of three parts: Tracking, Local Mapping and Loop closing. An overview can be seen in figure 3.1. Each part is ran on a different thread. The working of these three treads are explained further on.

All the optimization's in ORB-SLAM are done using the Levenberg-Marquardt algorithm of g2o [29]. The place recognition part contains a bag of words [16] description of the keyframes. The global map is created based on the covisibility graph. The covisibility graph has a node for each keyframe and edges between keyframes with at least 15 of the same map points. For the loop closure, also an essentials graph is created from the covisibility graph which consists of the same amount of nodes (one for each keyframe) but only the edges which have a high covisibility (more than 100 shared points).

Map Initialization

The map needs to be initialized based on the estimated pose between the first two frames. ORB-SLAM recognizes this as a difficult task and decides to compute two geometric models, one which works better with planar scenes, assuming for instance that it is looking at a wall, and one which works better with non-planar scenes. Only if one of these two result in a good enough configuration, a initial map is created.

The first step is to obtain the ORB matches as points x between the current frame F_c and the reference frame F_r: $x_c \leftrightarrow x_r$. The previous frame is used as the reference frame. If not enough matches are found (this threshold can be set as a setting in ORB-SLAM), it is tried again with the next frame set.

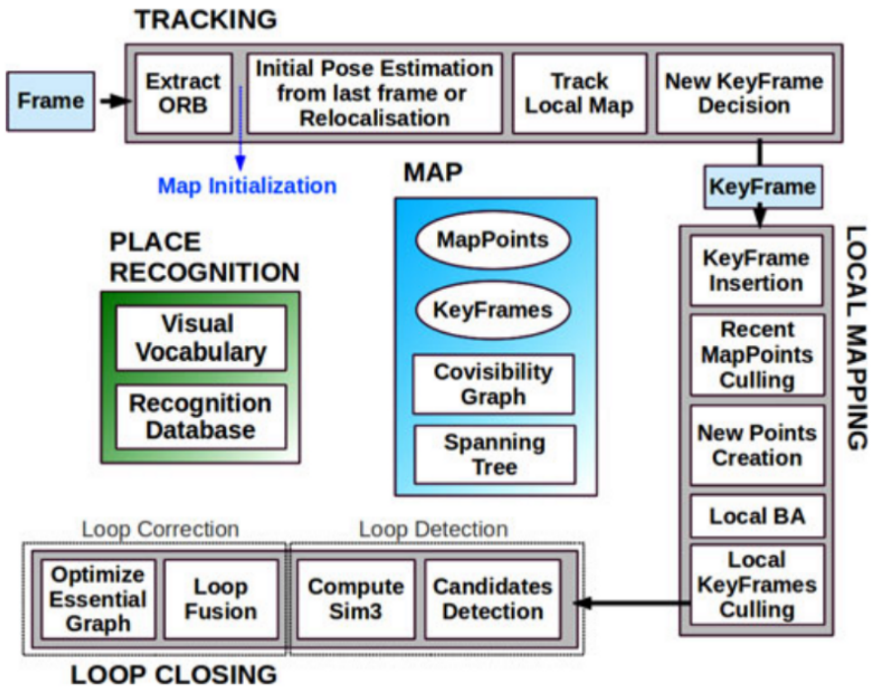


Figure 3.1: ORB-SLAM system overview which shows the three threads: Tracking, Local Mapping and Loop Closing, and there steps. Also the steps of the place recognition and contents of the map are shown. [39]

If enough ORB matches are found, initialization is attempted using two different models. One of these models is based on the homography and the other is based on the fundamental matrix. The homography H_{cr} and fundamental matrix F_{cr} are computed in parallel. The homography is computed using normalized DLT [23] with 4 points for a certain number of iterations. Equation 3.1 shows how the homography relates points in the reference frame to the current frame.

$$x_c = H_{cr} x_r \tag{3.1}$$

The fundamental matrix is computed using an eight-point algorithm [23] with 8 points for the same amount of iterations as when calculating the homography. Equation 3.2 shows how the fundamental matrix relates points in the reference frame to points in the current frame.

$$x_c^T F_{cr} x_r = 0 \tag{3.2}$$

For each iteration of each model M, a score S_M is calculated based on the symmetric transfer errors $d_{cr}^2(x_c^i, x_r^i, M)$ and $d_{rc}^2(x_c^i, x_r^i, M)$ [23]. How this score is set up can be seen in equation 3.3. The iteration of each model with the highest score is kept. To be able to deal with outliers, they set an outlier rejection threshold T_M for each model. They use for the homography model $T_M = 5.99$ and for the fundamental matrix model $T_M = 3.84$.

$$S_M = \sum_i (\rho_M(d_{cr}^2(x_c^i, x_r^i, M)) + \rho_M(d_{cr}^2(x_c^i, x_r^i, M))) \tag{3.3}$$

Where:

$$\rho_M(d^2) = \begin{cases} T_M - d^2 & \text{if } d^2 < T_M \\ 0 & \text{if } d^2 \geqq T_M \end{cases} \tag{3.4}$$

If no model with enough inliers is found, the process is restarted with the next frame. Otherwise the score of the best version of both models are compared according to equation 3.5.

$$R_H = \frac{S_H}{S_H + S_F} \tag{3.5}$$

In the case where $R_H >= 0.45$, a planar scene with low parallax is assumed. The homography model is used to retrieve eight motion hypotheses with the method of Faugeras and Lustman [14]. The eight hypotheses are directly triangulated. They check if there is one motion hypotheses with significantly more points with parallax from both cameras with low reprojection errors compared to the other motion hypotheses. If there is no solution that significantly stands out from the other motion hypotheses, the whole process is restarted with the next frame.

In the case where $R_H < 0.45$, a non-planar scene with high parallax is assumed. The fundamental matrix model is used to to create the essentials matrix with the calibration matrix K ($E_{rc} = K^T F_{rc} K$). This essential matrix is used with the singular value decomposition method [23] to obtain 4 motion hypotheses. These motion hypotheses are then directly triangulated. They check if there is one motion hypotheses with with significantly more points with parallax from both cameras with low reprojection errors compared to the other motion hypotheses. If there is no solution that significantly stands out from the other motion hypotheses, the whole process is restarted with the next frame.

Finally, with the best hypothesis, full bundle adjustment (see the appendix from [39] for details) is performed to obtain the initial map.

Tracking

The tracking part tries to find the pose difference between the previous frame and the new frame. It also decides if a new keyframe will be created or not. If it has lost track, it calls the place recognition module to attempt to find its place back in the global map. It takes the new frame, finds ORB point matches with the previous frame and then optimizes the pose of the new frame using motion-only bundle adjustment [39]. The keyframes with overlapping points are obtained by using the guess of the pose and the co-visibility graph of the past keyframes. The points of these keyframes are repojected to the new frame and matched. Using these matches, the pose is estimated more precisely. Then the choice is made to create a new keyframe or not.

The ORB points extracted from the current frame in the tracking part are used throughout the whole algorithm. They are extracted as FAST corners at eight scale levels with a scale factor of 1.2. The amount of corners extracted can be set in the settings, they use 1000 for image resolutions from 512x384 till 752x480 and more for larger images (for 1241x376 Kitti images they use 2000). The scale levels are divided into grid, trying to get 5 (or more if necessary to obtain the correct amount of features) FAST corners from each cell in the grids. This creates a evenly distribution of FAST corners over the frame. The resulting FAST corners are then described using the ORB descriptor.

Now that ORB points are extracted from the current frame, tracking can be done based on the last frame position. They initialize the pose of the current frame assuming that the camera has a constant velocity and do a guided search of map points in the last frame. If not enough corresponding map points are found, the search region is increased. When enough have been found, these matches are used to optimize the pose.

They create a local map as a set of keyframes. This set consists of the keyframes which have the same points as the current frame, together with all the neighbors to these keyframes. The keyframe with the most corresponding points with the current frame is set as the reference keyframe. Each point in this map is projected to the current image frame and multiple things are checked: if it lays within the image, if its viewing direction from the current frame is less than 60 degrees from the mean viewing direction of the point, if the distance from the camera center to the point is within the scale invariance region of the map point. Then these points are compared with the unmatched ORB features of the current frame to find extra matches. Then one last optimization is done using all the map points in the frame.

The only task left for the tracking thread is to decide if the frame should become a keyframe or not. ORB-SLAM has the strategy to create a lot of keyframes to be more robust to tricky camera movements (especially rotations). Later on they discard redundant keyframes through culling to be able to keep the computational cost of mapping low. The criteria for a frame to be accepted as a keyframe are:

- The frame must be at least 20 frames from the last global relocalization, to insure a good relocalization.
- The local mapping thread must be idle. If it has been more than 20 frames from the last keyframe insertion, the local mapping thread is stopped to be able to insert the keyframe and then restarted with the new keyframe.
- The frame must contain at least 50 points. This is necessary for good tracking.
- The amount of corresponding points between the frame and its reference keyframe (the keyframe with the most corresponding points) must be less than 90 % of the frames points.

Local Mapping
The local mapping thread takes the new keyframes and uses local bundle adjustment to optimize the local map around the keyframe. The still unmatched ORB points in the keyframe are compared to the points in the keyframes that it connects to in the covisibility graph. The matched points are culled to keep the best points. Also the keyframes are culled to remove the redundant keyframes, keeping the computational costs low. Culling is a process of removing a part of the set while keeping the amount of information in the set as high as possible.

When a new keyframe is created, the co-visibility graph is updated with a new node and the edges are set according to the shared points. The whole keyframe is then represented using bag of words. The new points in the keyframe must be visible in at least 3 of the 4 next keyframes and in at least 25 % of the keyframes it is predicted to appear in. Otherwise the point is culled, also if the point is at any time visible in less than 3 keyframes, it will be culled.

The new points that are about to be added to the map are triangulated. They are checked based on positive depth, parallax, reprojection error and scale consistency. The point is projected to connecting keyframes to find extra matches, after which it is added to the map. The just added keyframe with all its connected keyframes and their points is optimized with local bundle adjustment based on the other keyframes (which are not optimized, but remain fixed).

To keep the bundle adjustment computation low, the redundant keyframes are culled. They are discarded when more then 90 % of the points can be found in at least three other keyframes in the same or finer scale. They keep the keyframes that see the points with the most accuracy.

Loop Closing
Each time a new keyframe is added, the loop closing threads checks for loop closures. If a loop closure is found, a similarity transformation is set up to map out the drifts along the loop and the loop is aligned while merging duplicate points. Then a pose graph optimization over the similarity constraints is performed.

To obtain a set of loop closure candidates, the keyframe is represented as a bag of words vector. The similarity score between the keyframe and its neighbors in the co-visibility graph is computed. The lowest score is saved. For all other keyframes in the recognition database, the similarity score is computed. If the score of a keyframe in the recognition database is higher that the lowest score of the neighbors, the keyframe is considered a loop closure candidate. The candidate is only used if three consecutive possible candidates are found.

The similarity transform is calculated to obtain the drift error of the three translations, three rotations and one scale factor. If the similarity is supported by enough points, the similarity is optimized and more correspondences are searched. It is optimized again and again the amount of inliers is checked. If it is enough, the loop closure is accepted.

When a loop closure is accepted, all the transformation matrices of the keyframes are adjusted using the similarity transformation to align the loop. The duplicate points of the keyframes are fused and the edges in the covisibility graph are updated. The map needs to be optimized again, but only the part regarding the loop, so pose graph optimization is done over the essential graph.

Adding depth information
In the second version of ORB-SLAM, ORB-SLAM2 [40], the algorithm can also take stereo images or depth-maps as inputs. The stereo images or depth-maps are used to represent some of the ORB points as stereo points consisting of the coordinates in the left image and the vertical position in the (imaginary) rectified right image. These stereo points are then used as extra information in the optimization algorithms.

3.2. CNN setup

The CNN of CNN-SLAM [31] has been chosen to be used as the depth estimating CNN for this research. It is one of the best performing depth-estimating CNN's and it's implementation is openly available for use. It does not incorporate skip-connections as in [57], but it will prove that the performance of the CNN of CNN-SLAM outperforms depth estimating CNN's with skip connections such as [57]on the Kitti dataset. This network is larger than the other depth estimating CNN's, but this does not pose a problem as it fits on the GPU used for training. Fine-tuning will take longer than the other depth estimating CNN's.

The network architecture of the depth estimating CNN can be seen in figure 3.2. It consists of a combination of multiple convolutional layers, batch normalization layers, relu layers and a pooling layer. The

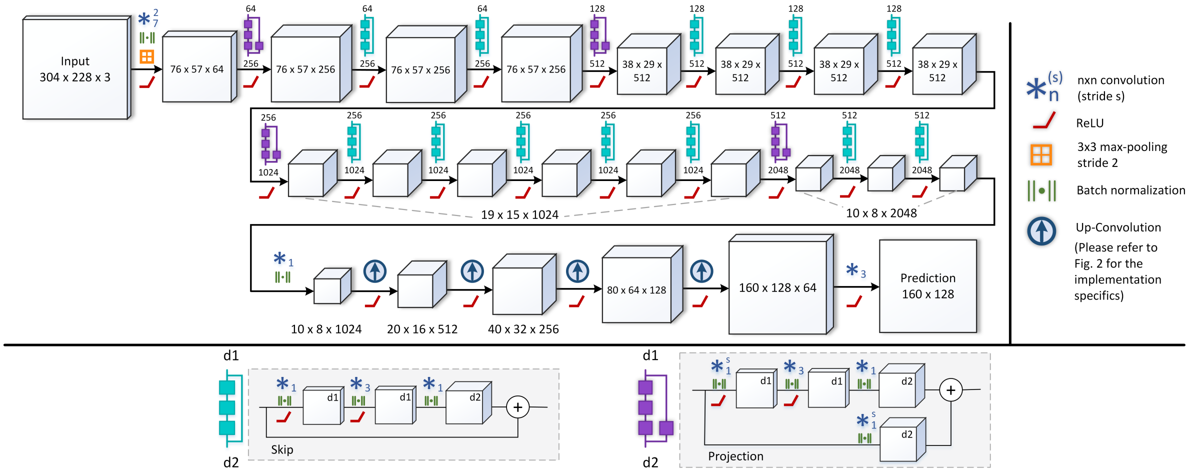


Figure 3.2: The network architecture of the depth estimating CNN used in this research, CNN-SLAM's CNN [31]. The which cubes show the intermediate sizes of the output from the previous set of layers for an example input image of 304x228x3. It is based on the ResNet-50 network [24], employing the same encoder. The decoder is designed by the authors of CNN-SLAM's CNN using their own designed up-sampling blocks, which can be seen in figure 3.3. The encoder consists of sequences of convolutional layers sometimes followed by batch normalization layers and ReLU layers. The same basic setup of layers repeats itself, always with a small local skip-connection. These are depicted in the architecture with the blue and purple icons, explained at the bottom of the image. They are either with layers in the skip connection (the purple icon) or without layers in the skip connection (the blue icon). The resulting output is about half the resolution of the input. This image is obtained from [31].

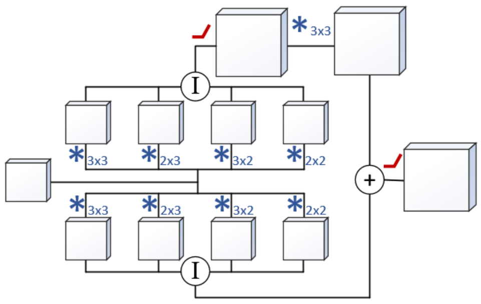


Figure 3.3: The up-sampling projection of the authors from [31], following the same icon meaning as in figure 3.2.

output depth-map is about half the size of the input image, and will be resized to the size of the input image afterwards. The network has an encoder-decoder style. The encoder 'encodes' the information into a feature representation, and the decoder decodes this feature representation into the depth image. It's encoder is the encoder from Resnet-50 [24] and the decoder is constructed by chaining multiple of the upsampling blocks designed by the authors of the CNN. The setup of the upsampling blocks can be seen in figure 3.3.

The authors have shared a predicting-only implementation of this network together with the set of learned parameters that resulted from their training on the NYU dataset [46]. This means that the fine-tuning setup needs to be created from the shared predicting setup.

For this research, the network has been setup in Tensorflow [1]. The authors have defined two different batch-normalization layers, one for during training and one for during predicting. In the implementation used in this research, one batch-normalization layer has been used during training and prediction. This improved the result.

For training, the network takes a folder with the images, a folder with the ground-truth depth-maps as images, a scale between the depth-map values and the depth in meters and the size of the images. The images and depth-maps are read using Tensorflow and divided into batches. The depth is scaled to meters. This is all done while keeping in mind that the image and depth-map pairs stay together. These batches are used to obtain the new input for the placeholders in the network each forward and backward pass. The same setup is used for predicting. The only difference is that the ground-truth depth is not used for training, but to evaluate the result of the forward pass.

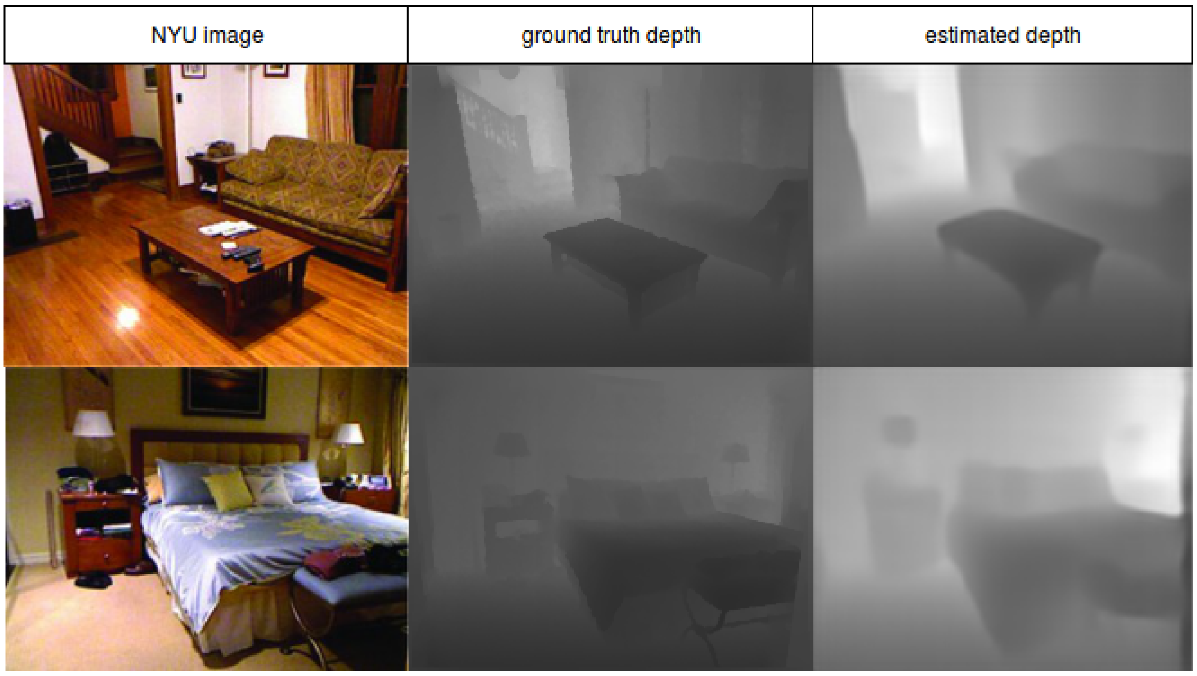


Figure 3.4: Examples of images from the NYU dataset, together with the ground-truth depth-map and the depth-map estimated by the CNN trained on the NYU dataset.

The loss function used by the authors of the CNN of CNN-SLAM [31] is the reverse Hubler function. The loss function implemented during this thesis is the L2 loss. The L2 loss is compared to the reverse Hubler loss in section 4.2. It will prove that the network fine-tunes better with the L2 loss then the reverse Hubler loss. See the determination of the hyper-parameters section, 3.4, for more details.

The network has been setup with the Adam optimizer [28] together with an exponentially decaying learning rate in an end-to-end matter. The network has not been trained from scratch with randomly initialized parameters. The parameters are initialized as the parameters from the original network trained on the indoor NYU dataset. This means that the network just needs to fine-tune the parameters to estimate on the new dataset instead of starting from scratch. This results in much faster training times.

The set of learned parameters that resulted from the authors training on the NYU dataset have been loaded into the network and then the network has predicted depth on images from the NYU dataset. Two examples of images together with the ground-truth depth and the estimate depth can be seen in figure 3.4.

The network needs images with a ground-truth depth-maps to fine-tune on. How these have been created can be found in sections 3.3. There are also multiple hyper-parameters that need to be chosen. The process of choosing this set of hyper-parameters for the fine-tuning can be found in section 3.4.

3.3. Preparing depth-maps for fine-tuning

Ground-truth depth-maps are needed for the fine-tuning of the network. However, obtaining these ground-truth depth-maps is not trivial. Otherwise the network would not be needed in the first place. The network is going to be fine-tuned on images from the Kitti dataset and on images from the WEpod, thus depth-maps of both these types of images are needed.

3.3.1. Kitti depth-maps

In the Kitti dataset, files containing the 64-plane lidar points are supplied together with the images. These points can be used to create the depth-maps for the undistorted images. The lidar points are in 3D space in the frame of the 64-plane lidar. These points are transformed to the 3D world frame of the camera using equation 3.6 where x is the point in the lidar frame, x' is the point in the camera frame, R the rotation matrix from the lidar frame to the camera frame and T is the translation vector from the lidar frame to the camera frame expressed in the lidar frame.

$$x' = R * (x - T) \tag{3.6}$$

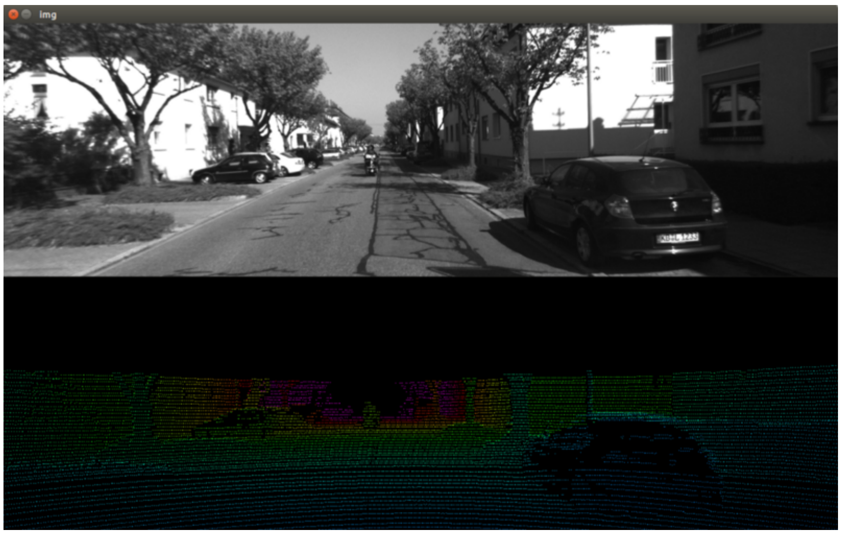


Figure 3.5: An example image and its sparse depth-map created by projecting 3D points from the 64-plane lidar to the image. The colormap flows from the lowest depth as blue and the furthest depth as purple.

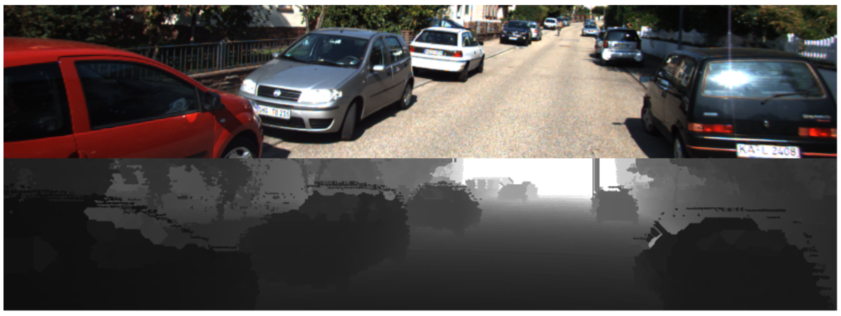


Figure 3.6: An example image and its dense depth-map created by the linearly interpolating projected 3D lidar points from a 64-plane lidar. The image has been cropped to a height of 228 to only contain the area that the lidar points cover.

All the points that lay behind the camera are removed. The points left over, those in front of the camera, are projected to the camera image frame using the intrinsic camera matrix K. This is done using equation 3.7 where x'' is the point in the image coordinates.

$$x'' = K * x' \tag{3.7}$$

The intrinsic camera matrix is created using the focal lengths in both directions (f_x and f_y) and the center of the camera in the image coordinates (c_x, c_y), see equation 3.8.

$$K = \begin{bmatrix} f_x & 0 & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{bmatrix} \tag{3.8}$$

The points behind the camera are filtered out because these would project falsely to the image frame. The points in front of the camera correctly project inside or outside the image. The points that lay inside the image are color code based on their depth in the camera world frame. An example of the resulting sparse depth-map is shown in figure 3.5.

This sparse depth-map is filled in to create a dense depth-map, where each pixels has a depth value. This is done by filling in the lidar points to horizontal lines. The pixels in between the points of the same line are filled in through linear interpolation. Then the space vertically between the lines is filled in through linear interpolation. The lidar points only reach to about the middle of the images. The top half without the lidar point coverage is cropped out, leaving an image height of 228 pixels. An example is shown in figure 3.6. To convert the depth-map to an image, the depth values are scaled such that a depth of 50 m maps to the maximum value of the image encoding (255). Any depth values greater then 50 m are set to 50 m.

3.3.2. WEpod depth-maps

Ground-truth depth data of WEpod images is needed for fine-tuning. The ground-truth depth-maps are also needed to evaluate the performance of the network. The WEpod itself has no way of creating depth-maps for its images with its current sensor setup. It is however possible to temporarily add a sensor to the WEpod that estimates the depth. This depth can then be transformed to the WEpod images. This is done using a Multisense S21 stereo camera setup.

The S21 has been placed inside the WEpod in such a way that its view overlaps as much as possible with the WEpod camera facing forwards. It is placed inside the WEpod with its left camera center about 30 cm above and about 20 cm behind the camera. Is is also placed facing straight forward, just as the WEpod camera. The exact rotation and translation values between the S21 and the WEpod camera will be estimated later on. The WEpod images can be matched to the corresponding S21 images using the timestamps of both the images.

With the S21 fixed to the WEpod in the described position, it can be used to record data. The S21 creates different outputs using both the left en right images. Two of these outputs have been recorded while the WE-pod drove around: the left image and the estimated depth-map of this left image. The estimated depth-maps are saved as grayscale images, with the depth thresholded at 50 m. Any depth larger then 50 m is set to 50 m. This recording has been done at 1 fps. Recording data any faster would not only put extra stress on the writing of the images to a disk, it would also not be necessary as the images would be very similar to each other.

The depth points need to be projected to the WEpod image to actually use the recorded depth. To do this, the rotation R and translation T between the S21 frame and the WEpod camera frame are necessary. Both R and the T have been estimated using stereo calibration. This is done by recording and undistorting multiple images with both the camera's of a checkerboard at multiple different positions. Each time, the calibration board is fully visible in both the images. In each of the images, the coordinates of the corners in the checker-board are searched using the OpenCV function 'findChessboardCorners()'. When all the coordinates of the corners in all the images of both camera's are combined, the R and T can be estimated. This is done through an optimization that projects all the points from one camera frame to the other camera frame and the other way around using an R and T. The reprojection error can be used to adjust R and T until the estimated R and T are optimized. This is done using the OpenCV function 'stereoCalibrate()'. The result can be sanity checked as R should be close to the identity matrix and T should be around the values estimated during placement of the S21 in the WEpod.

The points in the depth-map can be projected to the WEpod image by first projecting the point to the 3D S21 frame. This is done using the intrinsic camera matrix of the S21 left camera by multiplying the result of the OpenCV function 'undistortPoints()' with the depth to obtain the XYZ coordinates in the 3D S21 frame. These points in the 3D S21 frame (X^S) can then be transformed to the 3D WEpod frame (X^W) using the R and T:

$$X^W = R * (X^S - T) \tag{3.9}$$

The points in the 3D WEpod frame can be projected into the WEpod image frame using the intrinsic camera matrix in the OpenCV function 'projectPoints()'. Thus obtaining for this pixel in the image the depth value (the Z value of point in the 3D WEpod frame).

However, this cannot be done for every pixel in the WEpod image. The pixel might not be covered by the S21 camera, or the S21 could not estimate the depth of the point. The S21 calculates depth using stereo matching. It matches the points in the images of both its camera's and estimates the depth from the disparity of the point. If no match of a pixel could be found, due to for instance collusion or lack of texture, the depth of this pixel remains unknown. In the depth-map of the WEpod images, the unknown values are set to exactly 0. This because no depth of exactly 0 could exist in the image and this way these value can easily be filtered out during the fine-tuning of the CNN. During the fine-tuning, only the known values of the depth-map of the WEpod images are used.

3.4. Determining the hyper-parameters

There are multiple hyper-parameters in the network that need to be set for the fine-tuning. The exact value of these hyper-parameters influences the time needed for fine-tuning and the performance of the network. Setting a good combination of hyper-parameters can result in a fast and robust network that does not overfit.

Determining the value for the hyper-parameters is done by looking at the progress of the validation loss during the fine-tuning. The validation loss is the loss value calculated on a separate set of images that the

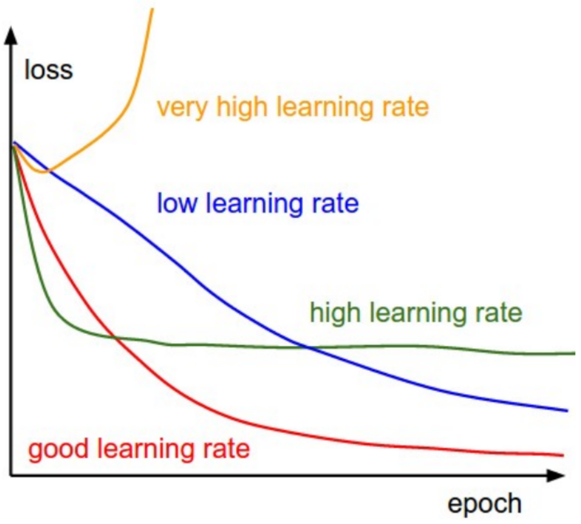


Figure 3.7: The change of loss values through epochs for different learning rates. [15]

network did not use for training, the validation set. This validation loss value indicates among others overfitting. When the validation loss starts to rise while the training loss lowers or stays the same, it is very possible that the network is starting to overfit to the training images.

The hyper-parameters in the network that have been experimented with during this thesis are the initial learning rate, the learning rate decay rate, the amount of epochs per decay and the loss function. The initial learning rate determines the value how much the weights are changed during fine-tuning. The decay rate together with the amount of epochs per decay determine how fast the learning rate goes down through epochs. The loss function chosen determines how different values of the error influence the weights.

3.4.1. Initial learning rate

The learning rate is used to change the value of the weights based on the gradient of the weights. The gradients of the weights are multiplied with the learning rate and then added to the weights. The initial learning rate is the learning rate set at the beginning of the fine-tuning. The AdamOptimizer lowers this learning rate through the epochs during fine-tuning.

A lower learning rate would mean that one would take smaller 'steps' during the optimization of the weights, increasing the time that it takes to learn. Lowering the learning rate increases the risk of the the optimization getting stuck in a local minimum instead of a global minimum. A higher learning rate results in less time needed to learn. However a learning rate that is to high results in the optimization not reaching the minimum of the error because it steps over the minimum.

Determining if a learning rate is to high or to low can be done by looking at the change of the validation loss during fine-tuning. A low learning rate results in a slowly descending validation loss, while a high learning rate results in a fast descending validation loss that can get stuck at a certain level. When the learning rate is way to high, the validation loss even starts to rise as the large steps force it to go towards worse weights. A learning rate that is just right results in a validation loss that drops fast but also keeps on dropping, reaching the lowest loss value. This is depicted in figure 3.7.

The original authors of [31] have trained the network on the NYU dataset using a training rate of 10^{-2}. As we only fine-tune it, a lower initial learning rate is suitable, so learning rates below the 10^{-2} will be examined. $\frac{1}{3} \times 10^{-3}$ will be examined because this value tends to be used a lot as an initial learning rate for training CNN's that work on images.

3.4.2. Learning rate decay rate

The decaying of the learning rate during the fine-tuning is done as a lower learning rate later on during the fine-tuning is preferable. This makes it possible to optimize closer towards the minimum of the error. A high learning rate in the beginning is preferable to quickly close in on the minimum of the error. The high learning rate in the beginning also helps skip over local minimums. Decaying the learning rate too fast results in the optimization getting stuck in a local minimum.

Decaying the learning rate is done in the AdamOptimizer [28] by setting two hyper-parameters: the amount of epochs per decay and the decay rate. The amount of epochs per decay determines when exactly the learning rate value drops. The decay rate is a value between 0 and 1 that determines the scale with which the learning rate is dropped. The AdamOptimizer is setup in such a way that each time the amount of epochs have passed, the learning rate is multiplied with the decay rate. This results in an exponential decay of the learning rate.

The network with only be fine-tuned, meaning that it will not process a lot of epochs. Choosing a high amount of epochs per decay would not make sense, as the learning rate would then hardly decay. Also does the resulting decay of the learning rate depend on the combination of the amount of epochs per decay and the decay rate. So it does not make sense to vary both these hyper-parameters a lot. The amount of epochs per decay will be hardly varied and the decay rate will be adjusted to the best setting with this amount of epochs.

3.4.3. Loss functions

The L2 loss function has been used as it is a fairly basic and often used loss function. The L2 loss function calculates the loss using the difference vector x between the vector of depth values predicted by the network y and the vector of ground-truth depth values y': $x_i = y'_i - y_i$ for each depth pixel i. The L2 loss function, shown in equation 3.10, calculates the loss as following where n is the amount of values in the vectors y and y'.

$$\text{L2 loss} = \frac{1}{n}\sqrt{\sum_{i=1}^{n} x_i^2} \tag{3.10}$$

The authors of the CNN of CNN-SLAM [31] used a different loss, the reverse Huber loss. Itis also called the berHu loss. This loss function is not used often and appears to be an interesting choice. Its original Huber loss is used more often. They do report an increase in performance with the berHu loss. The berHu loss function is a combination between the L1 loss function and the L2 loss function. The berHu loss is equal to the L1 loss (basic norm), but when this L1 loss exceeds a certain threshold c, it switches to the L2 loss. It is shown in equation 3.11.

$$\text{berHu loss} = \begin{cases} \frac{1}{n}\sum_{i=1}^{n}|x_i| & \sum_{i=1}^{n}|x_i| \le c \\ \\ \frac{1}{n}\sqrt{\sum_{i=1}^{n} x_i^2} & \sum_{i=1}^{n}|x_i| > c \end{cases} \tag{3.11}$$

For the value of the threshold c, the original authors use 20% of the maximum per-batch error, see equation 3.12.

$$c = \frac{1}{5} max_i(|x_i|) \tag{3.12}$$

A more clear representation of the the berHu loss function can be seen in figure 3.8. It shows the berHu loss for a 1 dimensional error x and $c = 10$. The authors claim that the berHu loss outperforms the L2 loss in their case because it penalizes the high error value more with the L2 loss, and because the L1 loss keeps the impact of the low error values high enough. They also claim that the L1 part of the berHu loss results in a higher derivatives for the low error value's than the L2, which results in a better performance of the berHu loss.

The fine-tuning of the network with the berHu loss will be compared with the fine-tuning of the network with the L2 loss. It would not make sense to compare the validation loss, as the way this loss is calculated changes. Therefore fine-tuning with both loss functions will be compared using the metrics for comparing depth maps defined in section 4.1.2.

3.5. Overall approach

The scientific community constantly tries to improve vision-based SLAM algorithms. One of the approaches to improve vision-based SLAM algorithms is adding depth information [40], [8], [55]. This depth information can be added to the inputs of ORB-SLAM as either sets of stereo images or depth-maps of the images. The improvement of the tracking accuracy of ORB-SLAM when ground-truth depth-maps of the images are added to the input is shown in the experiments.

Fortunately, the Kitti dataset has 64-plane lidar information which can be used to create depth-maps of the images. These are used as ground-truth depth-maps in this thesis. The process of creating these depth-maps from the 64-plane lidar points can be found in section 3.3. Unfortunately, the WEpod is not equipped

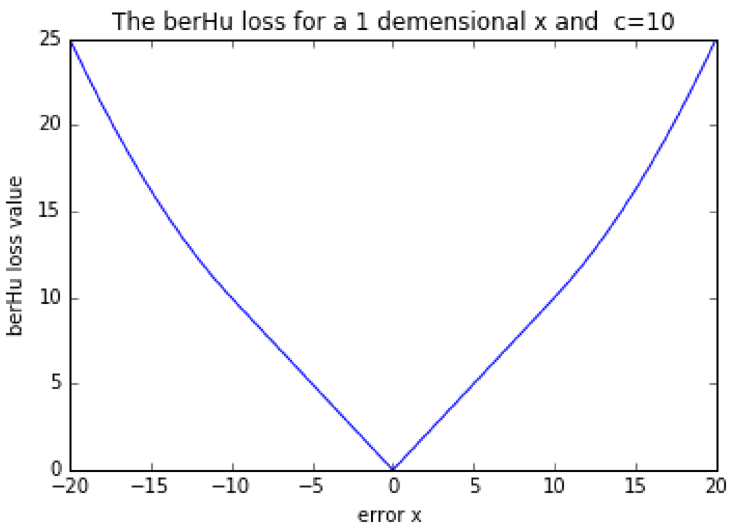


Figure 3.8: The berHu loss for a 1 dimensional error x and $c = 10$, within the range $[-10, 10]$ it is equal to the linear L1 loss, outside of this range it transfers to an quadratic L2 function.

with a 64-plane lidar. It is also not equipped with any other sensory setup that can used to create dense ground-truth depth-maps of its images. So it is not possible to try out ORB-SLAM with WEpod images together with their ground-truth depth-maps.

The improvement of the tracking performance of ORB-SLAM on WEpod images needs to be kept monocular. This means that the only input can be a stream of images from one camera. It needs to be kept monocular as the WEpod is not equipped with sensors able to estimate depth-maps for the images. There are however ways to estimate the depth-map of a WEpod images that uses only the image. The depth estimating convolutional neural network (CNN) part of CNN-SLAM [31] is an example. It is used in this thesis. This network estimates the depth-map of an input image, using only this image. It does need to be fine-tuned.

The CNN has originally been trained on the NYU dataset, which consists of images of indoor scenes. The network cannot estimate the depth in outdoor road scenery images as it has never seen any examples of these images. It can be trained to be able to estimate depth of outdoor road scenery images by training it on the same type of outdoor road scenery images. This training does not need to be done from scratch. The parameters of the network trained on the NYU dataset can be used as an initial starting point. The value of these parameters probably are already close the value they need to be to estimate the depth in the outdoor road images. The network only needs to be fine-tuned on the road scenery images. This is much faster then training the network from scratch.

As there is a way to create ground-truth depth-maps for Kitti images, the network can be fine-tuned on images from the Kitti dataset. The resulting depth-maps have been evaluated and compared to the results of the NYU dataset trained network, as can be seen in section 4.5. Before the fine-tuning on the Kitti dataset can be done, the hyper-parameters for fine-tuning need to be determined. The process of finding the values of these hyper-parameters is described in section 3.4 and the hyper-parameters themselves can be found in the experiments. These hyper-parameters are used throughout the whole thesis for each instance of fine-tuning.

The fine-tuned network can be added to ORB-SLAM, while keeping the whole setup monocular. This setup is depicted in figure 3.9. The images are fed into the depth estimating CNN. The resulting depth-map is fed into ORB-SLAM together with the image. ORB-SLAM estimates the trajectory, which it does in a non-monocular way. The whole setup together remains monocular. The resulting trajectory performance of this setup on Kitti sequences has been evaluated in section 4.6. The resulting trajectory accuracy can be compared to the trajectory accuracy of monocular ORB-SLAM and the trajectory accuracy of ORB-SLAM on images together with ground-truth depth-maps.

This same combination of CNN and ORB-SLAM can also be used on WEpod images. In section 4.7, the network fine-tuned on Kitti data is used to estimate the depth-maps of WEpod images. Networks trained on one dataset predicting images from other datasets do not always tend to perform well. The performance of

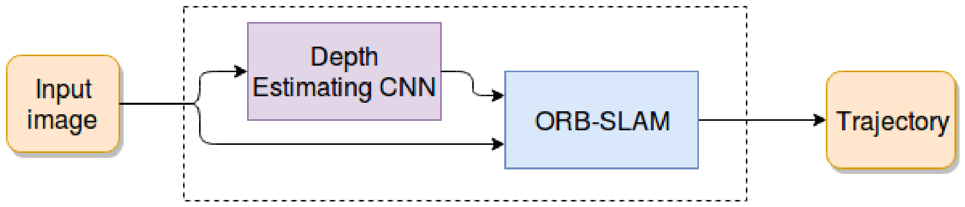


Figure 3.9: Diagram showing how the CNN can be added with a monocular vision-based SLAM algorithm (ORB-SLAM2 here as example), while keeping the whole system monocular. The images from the camera stream go into the CNN to estimate its depth-map. This depth-map is forwarded together with the image as inputs for ORB-SLAM2.

the Kitti fine-tuned network on the WEpod images however can be improved by making the images from the WEpod more similar to the Kitti images. This has been done by adjusting the height of the horizon in the images to the same height of the horizon in Kitti images.

It is also possible to fine-tune the network on WEpod images, see section 4.8. The ground-truth depth does need to be obtained, which can be done using the Multisense S21 stereo camera setup. The depth-map of the S21 can be projected to the WEpod images to obtain depth-maps. For details on how this is done, see section 3.3. These depth-maps do suffer from the same disadvantages as stereo depth, such as the inability to estimate the depth in case of occlusion or low-texture surfaces. Unfortunately can the images and depth-maps of the S21 stereo setup not be recorded fast enough with the equipment available to create ground-truth depth to run ORB-SLAM on.

There are two ways of estimating depth in WEpod images, with the CNN fine-tuned on Kitti images or fine-tuned on WEpod images. Both versions of the depth-estimating CNN have been added to ORB-SLAM, see section 4.9. The tracking performance of the ORB-SLAM on WEpod images together with the depth-maps estimated by this network can be compared to the tracking performance of ORB-SLAM on only WEpod images. This could show that the tracking performance monocular ORB-SLAM on WEpod images has been improved by adding a depth estimating CNN.

To summarize, the tracking accuracy of ORB-SLAM on WEpod images needs to be improved to be able to use it in the WEpod. The whole setup needs to be kept monocular. This could is by adding a depth estimating CNN to ORB-SLAM. The estimated depth-maps are fed together with the images into ORB-SLAM. ORB-SLAM can use this depth information to improve its tracking performance.

To investigate if adding depth information to ORB-SLAM indeed improves the tracking performance, images with ground-truth depth-maps are added to the inputs of ORB-SLAM. This is done with images from the Kitti dataset as it has a way of obtaining ground-truth depth. Also does ORB-SLAM actually initializes on only the Kitti images making it a better comparison. The ground-truth depth-maps are replaced with the output of the CNN to see how this changes the tracking performance ORB-SLAM.

Finally ORB-SLAM on WEpod images with the depth estimating CNN is compared to ORB-SLAM on only the WEpod images. This shows if the depth estimating CNN indeed improves the tracking performance. This would keep the whole setup monocular and applicable to be used on the WEpods.

4 Experiments & Results

The autonomous vehicle named WEpod is not capable of localizing itself robustly and accurately enough to ensure safe navigation through traffic. Monocular vision-based SLAM is investigated as a solution to improve its localization. The SLAM algorithm used is ORB-SLAM. It is investigated if the tracking accuracy of monocular ORB-SLAM is improved by adding a depth estimated CNN. This CNN can estimate the depth in images, which ORB-SLAM can use to more accurately and robustly perform SLAM.

This chapter explains the experiments done during this research and reports the results. The experiments can be divided into two types; evaluating ORB-SLAM's tracking accuracy on different inputs and fine-tuning plus evaluating the depth-estimating CNN. The first section of this chapter,4.1, explains the exact setup of both types of experiments together with the metrics used for evaluation of both the types of experiments.

Certain hyper-parameters need to be determined for fine-tuning, section 3.4. These hyper-parameters are determined by experimenting with different values and evaluating the performance. This process is further explained in section 3.4. The experiments and results are presented in section 4.2.

The other sections describe the goal of each experiment and presents the results. The ORB-SLAM with CNN setup is tested on the Kitti dataset. This gives the chance to investigate the actual difference in tracking accuracy better. The tracking accuracy of ORB-SLAM together with a depth-estimating CNN can be compared to monocular ORB-SLAM and ORB-SLAM with ground-truth depth-maps. The evaluation of monocular ORB-SLAM on Kitti images is presented in section 4.3. The evaluation of ORB-SLAM on Kitti images with ground-truth depth-maps is presented in section 4.4. The results of the fine-tuning of the CNN are shown in section 4.5. The performance of ORB-SLAM together with the CNN is evaluated in section 4.6.

There are two choices for fine-tuning the depth-estimating CNN for the WEpod images. First, using the CNN fine-tuned on the Kitti images to predict the depth in WEpod images, see section 4.5. The WEpod images need to be be altered to become more similar to the Kitti images. The process of altering the WEpod images and the performance of the depth-estimating CNN fine-tuned on Kitti images on WEpod images are explained in section 4.7.

The second option is to fine-tune the CNN on WEpod images. The depth-maps for fine-tuning are obtained with the S21 stereo sensor, as explained in section 3.3. The experiment concerning this fine-tuning is presented together with the results in section 4.8. Both these depth-estimating CNN's can be used together with ORB-SLAM. The tracking accuracy of ORB-SLAM with each CNN is evaluated in the final section of this chapter, section 4.9. With this evaluation, it can also be concluded if the addition of a depth-estimating CNN to ORB-SLAM indeed improves the tracking accuracy.

To summarize, the experiments are:

- Determine the hyper-parameters for fine-tuning.
- Evaluate the tracking accuracy of ORB-SLAM on only Kitti images.
- Evaluate the tracking accuracy of ORB-SLAM on Kitti images with ground-truth depth.
- Fine-tune the CNN on Kitti data.

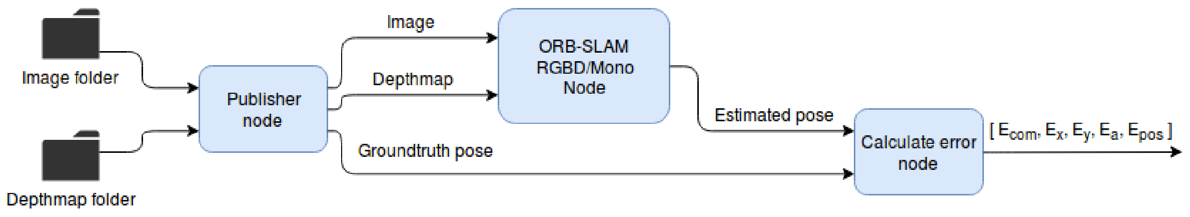


Figure 4.1: An overview of the ORB-SLAM setup in ROS, showing all the other nodes that communicate with the ORB-SLAM node. The publisher node sends the images and depthmaps to the ORB-SLAM node, which sends the estimated pose to the calculate error node. The publisher node also sends the groundtruth pose to the calculate error node, which calculates the error metrics.

- Evaluate the tracking accuracy of ORB-SLAM with the CNN on Kitti images.
- Predict depth in WEpod images with the CNN fine-tuned on Kitti data.
- Fine-tune the CNN on WEpod data.
- Evaluate the tracking accuracy of ORB-SLAM with the CNN fine-tuned on Kitti data and ORB-SLAM with the CNN fine-tuned on WEpod data.

4.1. Experimental setup

Two types of experiments are done multiple times under different circumstances. One is evaluating the tracking accuracy of ORB-SLAM on different amounts and types of inputs. The other is fine-tuning the depth-estimating CNN on different datasets and evaluating its depth-estimating performance.

The ORB-SLAM experimental setup is explained in section 4.1.1. Here the following is explained: all the changes that have been made to the code of ORB-SLAM2, how the algorithm's ROS node together with other self-written ROS nodes are setup, how the evaluation of the tracking accuracy takes place and which metrics are used for this evaluation.

The depth-estimating CNN experimental setup is explained in section 4.1.2. The settings during the fine-tuning are presented together with how the fine-tuning is done in tensorflow. The way of evaluating the depth estimating performance of the CNN is explained together with the metrics used for this evaluation.

4.1.1. ORB-SLAM experimental setup

The ORB-SLAM2 code comes with different executables that make use of the ORB-SLAM system in different ways. The implementation of ORB-SLAM in ROS is used. ROS is an environment that makes it possible to easily share information between different 'sub-programs' called nodes. Within ROS, it is easy to see exactly what information is shared to which node, useful for debugging. The sharing of information between ROS nodes is done through topics. A node can publish information to such a topic, and other nodes can subscribe to the topic to receive the information.

An overview of the ROS nodes used for the ORB-SLAM experiments can be seen in figure 4.1. Three nodes work together to calculate the error that evaluates the tracking accuracy of ORB-SLAM: the publish node, the ORB-SLAM node and the error node. Both the publish node and the error node have been created from scratch. The ORB-SLAM node is an adjusted version of either the RGBD or the Mono node shared in the code of ORB-SLAM2.

The publish node takes the arguments shown in table 4.1. The images and depth maps need to be sorted alphabetically in the folders. This is done by giving them ascending numbers as names. The node starts at the *s*'th image in the image folder. It publishes each image, corresponding depthmap and ground-truth pose to separate topics with a frequency of *h* Hz. When it has published the *f*'th image in the image folder, it stops publishing.

ORB-SLAM either needs to be run in monocular or rgbd mode. The corresponding ORB-SLAM node (Mono or RGBD) is used. The Mono node subscribes to the image topic and uses these images. The RGBD node subscribes to both the image and the depth-map topic and uses both the image and the depth-map to preform SLAM. The system code of ORB-SLAM has been altered for this purpose.

Argument	Description
h	The frequency with which the images needs to be published
image folder	The folder containing the images that need to be published.
depth folder	The folder containing the depth maps that need to be published.
pose file	A .txt file containing the ground-truth poses.
s	The node begins to publish the images from this number forward.
f	The node stops to publish the images from this number forward.

Table 4.1: The arguments for the publisher node with their description.

It is undesirable to let ORB-SLAM relocalize or use its loop closing capabilities as it would use its global map for these function. The tracking accuracy is being evaluated and not this global map. When ORB-SLAM loses track of where it is, it has failed for the rest of the path and no relocalizing is allowed. To prevent ORB-SLAM from relocalizing and finding loop closures two functions have been set to always return false: *LoopClosing::DetectLoop()* in line 103 of *LoopClosing.cc* and *Tracking::Relocalization()* in line 1341 of *Tracking.cc*.

The estimated pose of each frame needs to be extracted from the ORB-SLAM node. Code that does this is added to the *Tracking::GrabImageRGBD()* and *Tracking::GrabImageMonocular()* functions in *Tracking.cc*. These have been set up such that they both return the T_{wc} matrix. The T_{wc} matrix is the homogeneous transformation matrix from ORB-SLAM's world frame to the estimated camera frame. This T_{wc} matrix is returned through different functions until it is returned to the RGBD or Mono node. In the RGBD or Mono node the rotation matrix part of T_{wc} is calculated to quaternions ($[q_0, q_1, q_2, q_3]$) using ORB-SLAM's converter. These are then calculated to the Euler angles (roll: ϕ , pitch: θ, yaw: ψ) using equations 4.1, 4.2, 4.3.The euler angles are combined with the x, y and z values of the transformation part of T_{wc} into a pose message. This pose is defined in the world frame of ORB-SLAM. ORB-SLAM's world frame is the camera frame of the first image tracked. The pose message is sent to the calculate error node to be compared to the ground-truth pose.

$$\phi = tan2^{-1}\left(\frac{q_o q_1 + q_2 q_3}{1 - 2\left(q_1^2 + q_2^2\right)}\right) \tag{4.1}$$

$$\theta = sin^{-1}\left(2\left(q_0 q_2 - q_3 q_1\right)\right) \tag{4.2}$$

$$\psi = tan2^{-1}\left(\frac{2 * \left(q_0 q_3 + q_1 q_2\right)}{1 - 2\left(q_2^2 + q_3^2\right)}\right) \tag{4.3}$$

The node calculating the error subscribes to the ground-truth pose topic and the estimated pose topic. Each ground-truth pose that it obtains is stored inside a list for later use. When a estimated pose by ORB-SLAM is obtained, the corresponding ground-truth pose is obtained by matching the earlier set identification number of both the poses.

The monocular estimated poses have a free scaling parameter because monocular SLAM is unable to estimate this factor. Therefore, these poses are scaled such that the euclidean distance between the first two poses of the estimated trajectory and ground-truth trajectory are equal.

Both poses are transformed and projected to the same 2D frame. These poses consist of a longitudinal coordinate x, a lateral coordinate y and the yaw ψ. Projection to a 2D plane contains these three parameters, which are important for the localization of the WEpod. The height, roll and pitch are irrelevant as these don't describe the location on the road. Both the poses are saved into lists. The setup ends when the ORB-SLAM node and the publisher node have sent their last poses. Then the trajectory error can be calculated over these saved poses with the metrics described in the following paragraphs.

Four different metrics and a combination of these metrics are used to compare the tracking accuracy of ORB-SLAM. One of these metrics is the positional error E_{pos} (equation 4.8). It represents the average euclidean distance between each ground-truth pose and the corresponding estimated pose. This metric is very interesting for the WEpod. It shows how much your estimation is off after for instance a corner. For the comparison of the trajectory accuracy this metric is not interesting. The metric is heavily depend on the position in time when an error is made. A rotational error made in the beginning has a far greater influence than a rotational error made in the end.

Therefore, three other metrics are used; the longitudinal error E_x (equation 4.5), the lateral error E_y (equation 4.6) and the rotational error E_a (equation 4.7). These metrics compare the average velocity (the difference over the last time step) of the poses. The rotational and translational performances are evaluated separately as vision-based SLAM methods tend to perform very different for rotational and translation trajectories. The translational error has also been split into a longitudinal error E_x and a lateral error E_y to be able to investigate the difference in performance of both movements separately.

Finally there is also the combined error E_{com} (equation 4.4). It represents one value indicating the increase or decrease of performance. The combined error consists of a weighted combination of the longitudinal error, the lateral error and the rotational error. The positional error is left out on purpose as it is less interesting for the comparison of the tracking accuracy. The weights normalizes the different errors and scales the errors based on their importance's. Each error is normalized by dividing it by its average value over all the experiments. The weight on the longitudinal error is multiplied by two and the the weight on the rotational and lateral error is multiplied by four. This is done because during driving an error in the longitudinal direction is much less severe than a lateral or rotational error.

The rotational error takes the angles in degrees and the others take the position in meters. All the metrics are scaled by the fraction of poses ORB-SLAM has estimated. For instance if ORB-SLAM initialized after 13 frames, lost tracking after 126 frames and the total path consisted of 150 frames, the metrics are scaled by the total amount of frames N divided by the amount of estimated frames n: $\frac{150}{113} = 1.33$. If less than 1% of the frames have been estimated by ORB-SLAM, the trajectory prediction has been seen as failed and all the errors are set to a their set maximum value of 100. No metric can exceed a value of 100, resulting in a combined error of 1000.

$$E_{com} = \frac{2.0}{0.742}E_x + \frac{4.0}{0.101}E_y + \frac{4.0}{1.255}E_a \tag{4.4}$$

$$E_x = \frac{N}{n^2}\sum_{i=1}^{n}\left|\left(x_i^g - x_{i-1}^g\right) - \left(x_i^o - x_{i-1}^o\right)\right| \tag{4.5}$$

$$E_y = \frac{N}{n^2}\sum_{i=1}^{n}\left|\left(y_i^g - y_{i-1}^g\right) - \left(y_i^o - y_{i-1}^o\right)\right| \tag{4.6}$$

$$E_a = \frac{N}{n^2}\sum_{i=1}^{n}\left|\left(\psi_i^g - \psi_{i-1}^g\right) - \left(\psi_0^o - \psi_{-1}^o\right)\right| \tag{4.7}$$

$$E_{pos} = \frac{N}{n^2}\sum_{i=0}^{n}\sqrt{\left(x_i^g - x_i^o\right)^2 + \left(y_i^g - y_i^o\right)^2} \tag{4.8}$$

The setup is used on images from two datasets; the Kitti dataset and the WEpod dataset. Five different sequences have been taken from each to be used as the input image streams. This is done to compare the tracking performance when taking different corners and straight roads. A top view of the paths of the five sequences from the Kitti dataset can be seen in figure 4.2 and from the WEpod dataset can be found in figure 4.3.

The results of ORB-SLAM are not very consistent. When running ORB-SLAM on the same sequence of images under the same circumstances, it can perform very good one time and terrible the next time. This is because ORB-SLAM uses a lot of optimization methods such as bundle adjustment. Comparing the results of one run is not enough to be certain that improvement has occurred.

Therefore, each sequence is run 20 times and the 95 % confidence interval of the mean is used for comparison. The mean μ itself is calculated by taking the average error over N runs ($N = 20$). This can be seen in equation 4.9, where E_i is the error for run i. The standard deviation σ is also needed and calculated as in equation 4.10. The 95 % confidence interval I_E of the mean is calculated using the $Z_{0.95} = 1.960$ value (equation 4.11).

$$\mu_E = \frac{1}{N}\sum_{i=0}^{N} E_i \tag{4.9}$$

$$\sigma_E = \sqrt{\frac{1}{N}\sum_{i=0}^{N}\left(\left|E_i - \mu_E\right|^2\right)} \tag{4.10}$$

$$I_E = \left[\mu_E - Z_{0.95}\frac{\sigma_E}{\sqrt{N}}\,,\, \mu_E + Z_{0.95}\frac{\sigma_E}{\sqrt{N}}\right] \tag{4.11}$$

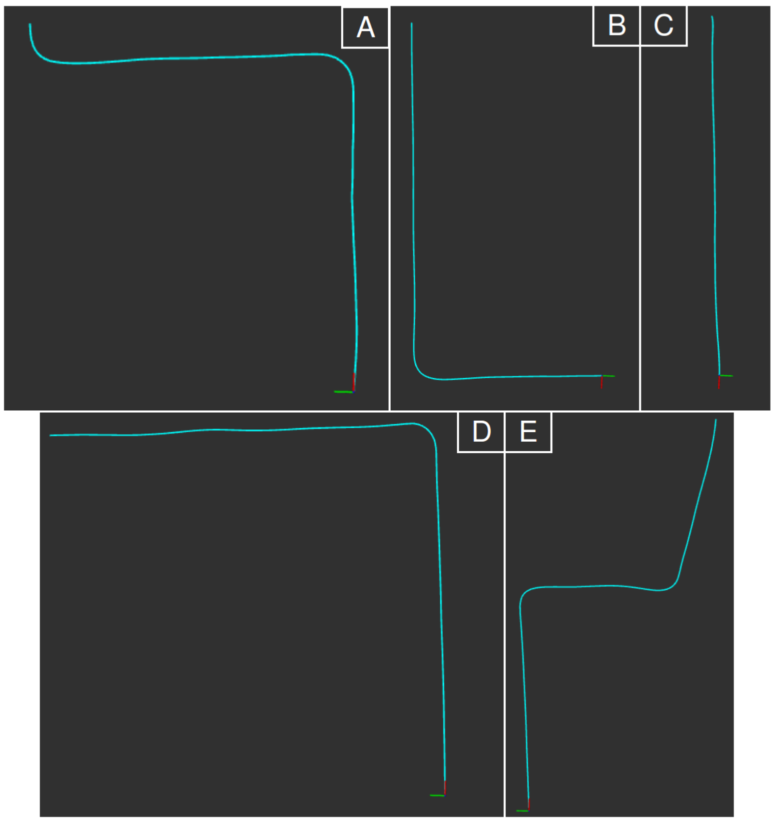


Figure 4.2: The five top views of the sequences used from the Kitti dataset (from their sequence 00). Each sequence start at the red and green axis (bottom of the images). Their is a straight sequence (C), a left turn (D), a right turn (B), a left then right turn (A) and a right then left turn (E).

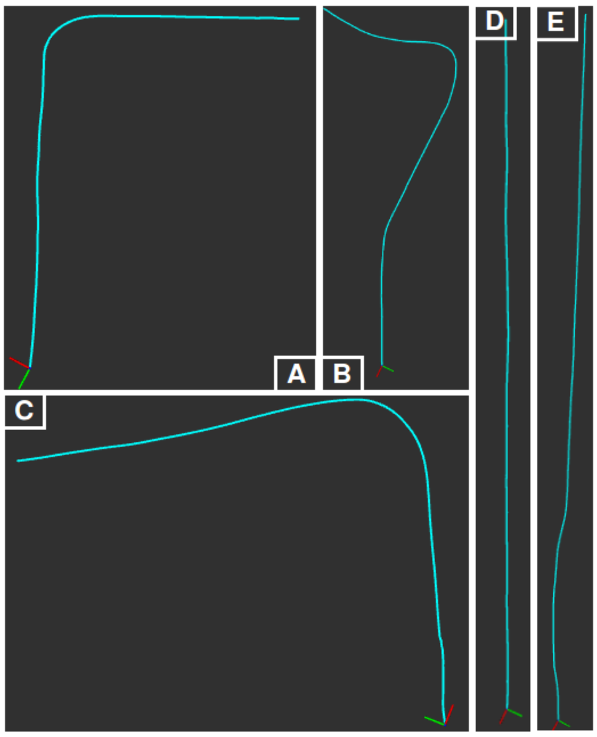


Figure 4.3: The five top views of the sequences used from the WEpod dataset. Each sequence start at the red and green axis (bottom of the images). Their is a right turn (A), a right then left turn (B), a left turn (C), a totally straight path (D) and a straight path with a parked truck passing (E).

4.1.2. Depth-estimating CNN experimental setup

The depth estimating CNN of CNN-SLAM [31] is used in this research. A detailed description of the network can be found in section 3.2. The authors have shared an forward passing only implementation of this network trained on the NYU dataset [46] in Tensorflow [1]. A new implementation with the original shared implantation as inspirational source. The network is implemented in Tensorflow.

The only difference between the layers shared by [31] and the layers used in this research is the batch normalization layers. The authors used two different versions of these layers; one for training and one for predicting. This however proved to worsen the performance. The network in this research only uses the training version of the batch normalization layer. The other layers in the network are all kept the same.

The hyper-parameters have been set as described in section 4.2. The dropout parameter did not make much difference and has not been experimented with in section 4.2. The dropout parameter has been set to 0.5, as this is also the value the original authors used. Their is only one dropout layer in the end of the network, thus the exact value of the dropout rate does not have a great influence. It is good to not just set it to 1, as this could result in the network overfitting.

The ground-truth images from the WEpod are created using a stereo setup and thus do not contain the depth for each pixel, section 3.3. The unknown pixels have been given a depth of 0, as nothing else can have a depth of exactly 0. The ground-truth depth is also capped at 50 m. Every depth above 50 m has been set to 50 m. This has been done because the depthmaps are saved as images, meaning that the depth is saved scaled to an integer in the 0 to 255 range. This means that a maximum depth must be defined and set to scale to the 255 pixel value.

If this maximum depth is to large, a lot of accuracy is lost because the scaled value is rounded to the nearest integer. If the maximum depth is to small, the network would not learn to estimate the depth beyond this range and the resulting estimated depthmaps would be useless. The maximum depth value of 50 m is assumed to be large enough to result in useful depthmaps (during driving, the most useful information is within 50 m for the WEpod) and small enough to keep a good accuracy.

The loss function used is the L2 loss function, section 4.2. It is adjusted to fit the ground-truth depth images explained above. First the output of the network is scaled bi-linearly to the size of the input image and ground-truth depthmap. This implementation of the loss function filters out the pixels with a groundtruth depth values of 50 m and smaller than 1m. The pixels left over are used to calculate the L2 loss.

The image and ground-truth depthmap combinations used for the fine-tuning of the network have been split up randomly into three groups: The training group consists of 85 % of the images, the validation group consists of 10 % and the testing group consists of 5 % . The training group is used to train on. The validation group is a group of images the network has never seen before. The performance of the network on the validation group is used to choose the hyper-parameters with (section 4.2). The testing group consists of images the network never has seen before and which not have been used to shape the hyper-parameters with. The testing group is used to determine the performance of the network.

7481 Kitti images from different sequences have been divided over the three groups for the fine-tuning. 642 WEpod images have also been divided into the three groups for fine-tuning. The fine-tuning has been done on a Nvidia geforce GTX 1080 Ti with 12 Gb memory. Fine-tuning was done with batch-sizes of 8 for the Kitti images and 4 for the WEpod images. Using larger batch-sizes was not possible due to the size of the memory space.

The depth estimating performance is determined using four metrics. Three of the metrics are also used by the original authors of the network [31] and others [5] [6] [30] [32] [34]. The four metrics are: $RMSE$ (equation 4.12), $RMSE(log)$ (equation 4.13), the absolute relative difference (ard) (equation 4.14) and the squared relative difference (srd) (equation 4.15). The first three are from [31] and the last one is from [6]. They have been calculated over all the pixels where the ground-truth depth value is greater then 1 and smaller than 50. The estimated depthmap has first been resized bi-linearly to the same size as the input images and ground-truth depthmaps. In the metric equations n is the amount of pixels, d_i^* is the groundtruth depth value of pixel i and d_i is the estimated depth value of the pixel i.

$$RMSE = \sqrt{\frac{1}{n}\sum_{i=0}^{n}\left(\left(d_i^* - d_i\right)^2\right)} \tag{4.12}$$

$$ard = \frac{1}{n}\sum_{i=0}^{n}\left(\frac{|d_i^* - d_i|}{|d_i^*|}\right) \tag{4.14}$$

$$RMSE(log) = \sqrt{\frac{1}{n}\sum_{i=0}^{n}\left(\left(\log\left(d_i^*\right) - \log\left(d_i\right)\right)^2\right)} \tag{4.13}$$

$$srd = \frac{1}{n}\sum_{i=0}^{n}\left(\frac{|d_i^* - d_i|^2}{|d_i^*|}\right) \tag{4.15}$$

4.2. Determining the hyper-parameters

Before the depth-estimating CNN can be fine-tuned, its hyper-parameters need to be determined. An explanation of the hyper-parameters that need to be set can be found in section 3.4. The main goal of the following experiments is to find a set of hyper-parameters that ensures that the network fine-tunes in a fast and robust way without overfitting. This is done by comparing the progress of the validation loss during fine-tuning with different values for of the hyper-parameters. The other hyper-parameters are kept constant, making sure that the the difference of the validation loss is due to the change of the one hyper parameter.

The hyper-parameters that have been experimented with are; the initial learning rate, the learning rate decay rate, the amount of epochs per decay and the loss function. The initial learning rate determines how much the weights are changed for the first epoch during fine-tuning. The decay rate together with the amount of epochs per decay determines how fast the learning rate goes down through epochs being processed. The loss function chosen determines how the error that the network makes influences its weights.

4.2.1. Initial learning rate

Different learning rates have been examined: 10^{-2}, $\frac{1}{3} \times 10^{-3}$, 10^{-3}, 10^{-4}, 10^{-5}, 5×10^{-6} and 10^{-6} . The network has been setup for fine-tuning according to section 4.1.2. The network has been fine-tuned for 5 epochs. This is long enough to show the difference between the different learning rates. The amount of epochs per decay is set to 1 and the decay rate is set to 0.9, which are often used values. The loss used here is the basic L2 loss, further described in section 3.4.3. The L2 loss on the validation dataset is shown in figure 4.4.

Increasing the learning rate from 10^{-6} till 10^{-4} decreases the time needed for the fine-tuning, as the validation loss goes down faster (figure 4.4). When increasing the learning rate higher then 10^{-4}, the validation loss does drop faster, but it also stops dropping much earlier. This results in a higher loss after 5 epochs. The initial learning rate value of 10^{-4} is considered to be the most suitable and will be used from here on.

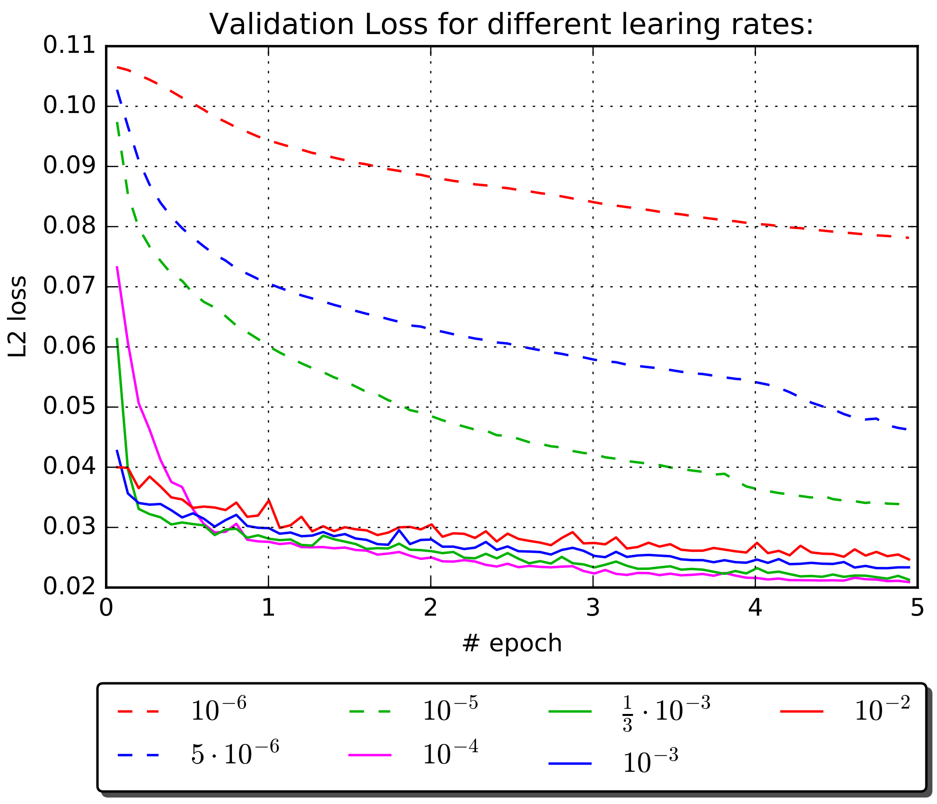


Figure 4.4: The validation L2 Loss of different learning rates (10^{-2}, $\frac{1}{3} \times 10^{-3}$, 10^{-3}, 10^{-4}, 10^{-5}, 5×10^{-6} and 10^{-6}). The model has been trained for 5 epochs with these different learning rates, an amount of epochs per decay of 1 and a decay rate of 0.9.

4.2.2. Learning rate decay rate

The network will only be fine-tuned, meaning that it will not process a lot of epochs. Choosing to decay the learning rate after a high number of epochs would not make sense. The learning rate would then hardly decay. The speed of the learning rate decay depends on the combination of the scale that decays the learning rate each time (decay rate) and the amount of epochs in between the decay steps (epochs per decay). Two values of the amount of epochs per decay have been examined: 0.5 and 1. For the decay rate, a wider spectrum has been chosen to examine if quickly lowering the learning rate could prove positive. The decay rates examined are: 0.2, 0.5, 0.7 and 0.9. The resulting validation L2 loss through the epochs for all the 8 possible combinations can be seen in figure 4.5.

Some combinations get stuck in certain local minimums, figure 4.5. These are the combinations that lower the learning rate very fast, such a decay rate of 0.2 or 0.5. It appears that the combinations that lower the learning rate the slowest result in the lowest validation loss. These are thus the best to use. Between these combinations, no apparent difference can be seen, so they are examined into further detail for more epochs.

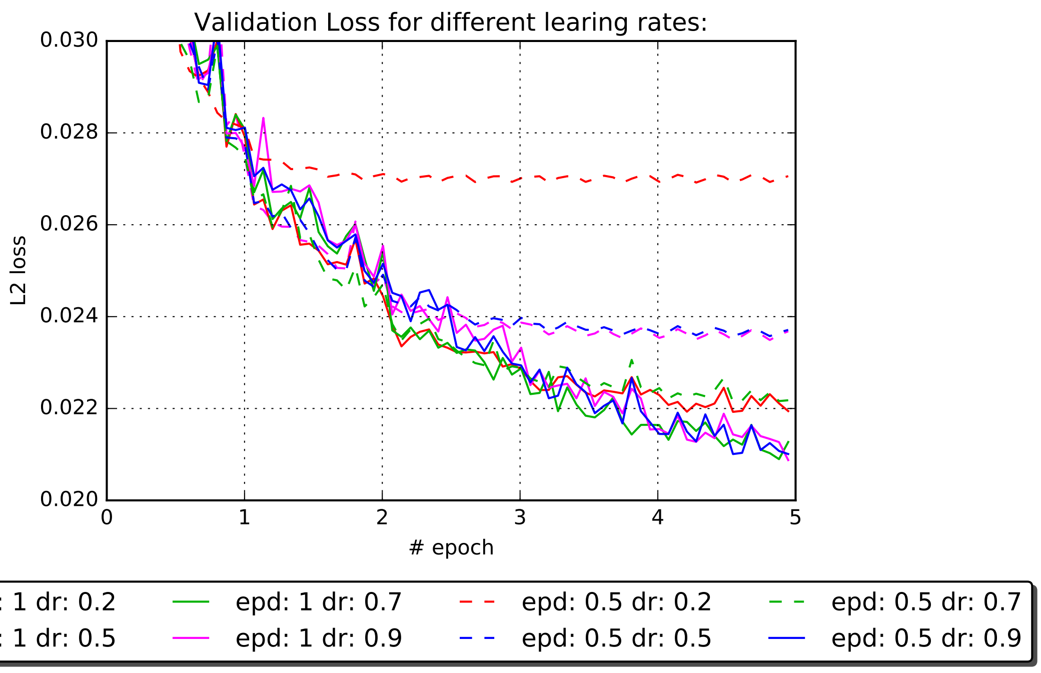


Figure 4.5: The validation L2 Loss of different decay rates (dr: 0.2, 0.5, 0.7 and 0.9) combined with different amount of epochs per decay (epd: 1 and 0.5). The model has been trained for 5 epochs with these different decay rates and a learning rate lr of 10^{-4}.

The amount of epochs per decay has been set to 1 and no other values are examined. From figure 4.5 it can be concluded that the combinations that result in about the same decay of the learning rate result in the same validation loss through the epochs. As the amount of epochs per decay is kept constant, the decay rate is varied. The decay rates examined this time are: 0.8, 0.85, 0.9, 0.95 and 0.99. To see the difference after more epochs, the network is now fine-tuned for 25 epochs. The resulting validation L2 loss through the epochs for these different decay rates can be seen in figure 4.6.

It can be seen in figure 4.6 that the difference in validation loss between the different decay rates only becomes visible after about 10 epochs. The validation loss eventually stays at a certain value for each decay rate. The higher the decay rate, the slower the learning rate decays and the lower the eventual value is. Only between 0.95 and 0.99 is no real difference. They both stagnate at the same validation loss. Based on these results, the decay rate of 0.95 together with a 1 epoch per decay rate is used from now on.

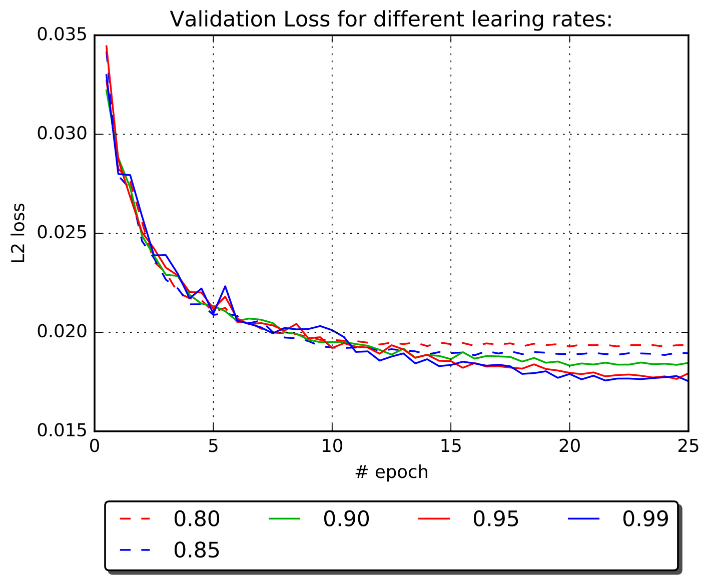


Figure 4.6: The validation L2 Loss of different decay rates (dr: 0.8, 0.85, 0.90, 0.95 and 0.99). The model has been trained for 25 epochs with these different decay rates, an epoch per decay of 1 and a learning rate lr of 10^{-4}.

loss	lr	epd	dr	# epochs	RMSE	log RMSE	abs rel diff	squared rel diff
L2	0.0001	1	0.95	25	3.97	0.22	0.17	1.05
berHu	0.0001	1	0.95	25	4.50	0.25	0.20	1.38
berHu	0.01	7	0.9	25	5.16	0.29	0.25	2.07
L2	0.0001	1	0.95	50	3.98	0.22	0.16	1.03

Table 4.2: Comparison of the performance of the network fine-tuned on the Kitti dataset with the L2 loss and the hyper-parameters found above for 25 epochs, the berHu loss and the hyper parameter found above for 25 epochs, the berHu loss and the hyper-parameters from [31] for 25 epochs and the L2 loss with the hyper-parameters found above for 50 epochs

4.2.3. Loss function

The fine-tuning the network with the reverse Huber (berHu) loss is compared with fine-tuning of the network with the L2 loss. The evaluation is done using the metrics for comparing depth-maps defined in section 4.1.2. The metrics are averaged over all the validation images. Four different settings are used for comparison. The first one uses the L2 loss, the rest of the hyper-parameters set to the values found above and for 25 epochs. The second one is using the berHu loss with the rest of the hyper-parameters set to the values found above and for 25 epochs. Then also the berHu loss with the hyper-parameters set to the same values as the author used is fine-tuned for 25 epochs. Finally also the best performing one is fine-tuned for 50 epoch to test if fine-tuning for 25 epochs is enough. The results can be seen in table 4.2.

The L2 loss results in better performance than the berHu loss. Even though this is opposite of the findings of the authors of the CNN. This could be because of the different datasets consisting of images from different sceneries. The NYU dataset is an indoor dataset with much lower depth values. The Kitti dataset is an outdoor dataset that consist of much larger depth values. The table also shows that the settings found above result in better performance than the settings from the authors of the network. It shows that fine-tuning for more than 25 epochs does not result in better performance, so it is not needed to fine-tune the network any further after 25 epochs. A summery of the hyper-parameters used for this thesis can be found in table 4.3.

Hyper parameter	Value
Learning rate	10^{-4}
Amount of epochs per decay	1
Decay rate	0.95
Amount of epochs	25

Table 4.3: The summery of the hyper-parameters used for fine-tuning the depth estimating network.

4.3. Monocular ORB-SLAM on only Kitti images

The performance of monocular ORB-SLAM on Kitti images is investigated in this experiment using the metrics described in section 4.1.1. First will be recapped why ORB-SLAM on Kitti images is investigated while the goal is to improve ORB-SLAM on WEpod images.

Monocular ORB-SLAM on WEpod images does not initialize and no trajectory is estimated. This results in undefined values of the tracking accuracy metrics. If ORB-SLAM with the depth estimating CNN would initialize, its tracking accuracy would be better then the monocular attempts. This would mean that the tracking accuracy has improved. However, it would only prove that the initialization problem is solved. If and how much the actual tracking performance has improved remains unanswered. Therefore, it is desirable to compare the tracking accuracy of ORB-SLAM with and without the estimated depth-maps on a different dataset. Such a dataset is the Kitti dataset on which ORB-SLAM does initialize and thus its tracking accuracy can be compared. This experiment sets the base-line to compare the other tracking accuracy metrics of ORB-SLAM on Kitti data with. ORB-SLAM is run according to the experimental setup described in section 4.1.1 on the five selected sequences of Kitti images, 20 times per sequence.

4.3.1. Results

The tracking accuracy of monocular ORB-SLAM on the Kitti images is shown in table 4.4. The mean μ and standard deviation σ of each error metric on each sequence are presented. A somewhat more clear representation for comparison is presented in the sub-figures of figure 4.7. In these figures the mean and 95 % certainty interval is presented for error metric and each sequences of Kitti images.

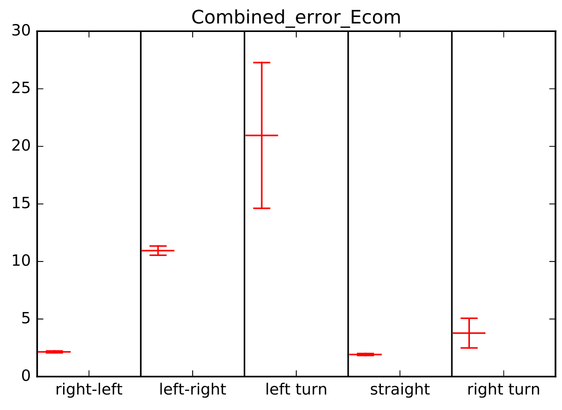
The results between the different sequences vary a lot, see table 4.4. For instance, the sequence containing a left turn has relative very high errors on each metric. The straight sequence containing no turns performs better then the others on each metric. This is more clearly visible in the sub-figures of figure 4.7. The 'left turn' sequence not only has the highest error of the sequences, but also has the largest 95 % certainty interval of the mean. This indicates that the performance of ORB-SLAM on the 'left turn' sequence of the Kitti images varied a lot throughout the runs.

This relatively bad and unreliable result of the 'left turn' sequence is present in the longitudinal error, E_x. Also the 'right turn' sequence resulted in a high variation in performance in the longitudinal error. In the yaw error E_a, not only the 'left turn' sequence performs badly with a high variance, the 'left-right' sequence also performs relatively badly. However, it does not have a high variance. In the combined error, E_{com}, the difference between the 'left turn' sequence and the others is also visible. The 'left-right' sequence stands out because of its poor performance due to poor performance in E_a. Another difference that stands out is between the longitudinal error E_x and the lateral error E_y. The error in the lateral direction is smaller than the error in the longitudinal direction for each sequence. Not only is the mean of the lateral error substantially smaller, also is the variance is smaller.

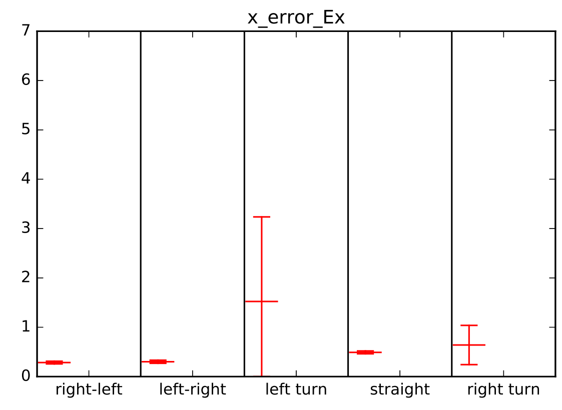
The base-line has been set. These results can be used to show the change in performance when adding depth-information to the inputs. These results also already show the difference in ORB-SLAM's tracking accuracy between different sequences of Kitti images. The next step is to investigate how the addition of ground-truth depth-maps to the inputs of ORB-SLAM affect the tracking accuracy.

Error metric		**Sequence**				
		Right-Left	Left-Right	Left	Straight	Right
E_{com}	μ	2.145	10.941	20.946	1.917	3.776
	σ	0.184	0.912	14.443	0.19	2.94
E_x	μ	0.284	0.301	1.522	0.491	0.640
	σ	0.063	0.072	3.910	0.066	0.907
E_y	μ	0.022	0.046	0.094	$9{\cdot}10^{-3}$	0.040
	σ	$9{\cdot}10^{-3}$	$3{\cdot}10^{-3}$	0.223	$1{\cdot}10^{-3}$	0.041
E_a	μ	0.167	2.615	4.127	0.070	0.146
	σ	0.020	0.278	1.416	0.014	$4{\cdot}10^{-3}$
E_{pos}	μ	37.348	40.672	52.758	35.391	40.126
	σ	6.513	2.685	11.190	0.765	13.750

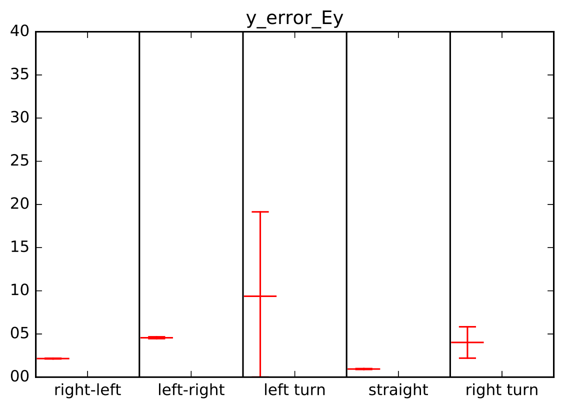
Table 4.4: The tracking accuracy of monocular ORB-SLAM on the five different sequences from the Kitti image set. The error metrics and how there mean μ and standard deviation σ are described in section 4.1.1.



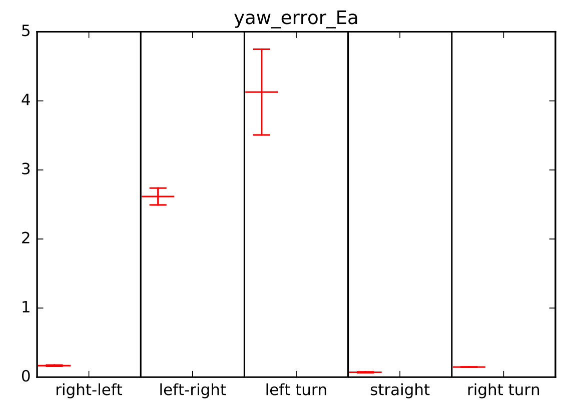
(a) The combined error E_{com}



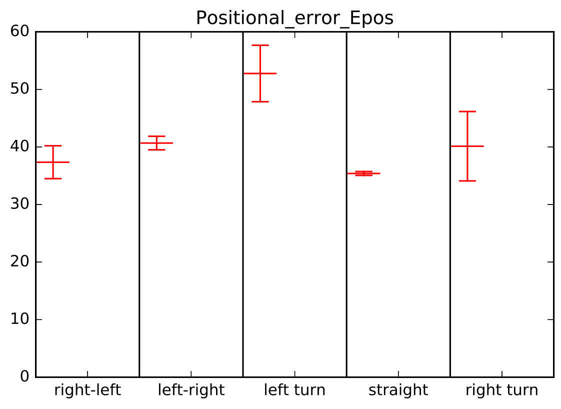
(b) The longitudinal error E_x



(c) The lateral error E_y



(d) The yaw error E_a



(e) The positional error E_{pos}

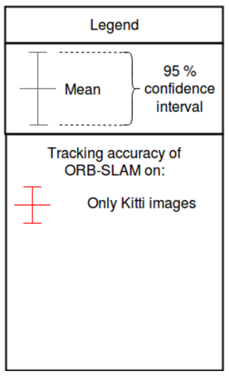


Figure 4.7: The tracking accuracy of monocular ORB-SLAM on Kitti images. The average value and 95 % certainty interval of each metric is shown for each sequence.

4.4. ORB-SLAM on KITTI images with lidar depthmaps

This experiment verifies that adding depth information to ORB-SLAM improves the tracking accuracy. This is necessary for a fair comparison. It cannot be done on WEpod data, as no ground-truth depth-maps applicable for use with ORB-SLAM are currently available. For the Kitti dataset however, ground-truth depth-maps can be create with the 64-plane lidar data, as done in section 3.3.

The tracking accuracy of ORB-SLAM on images together with their ground-truth depth-maps is evaluated. It gives the possibility to compare its results it with others. For instance, comparing the tracking accuracy of ORB-SLAM with estimated and with ground-truth depth-maps. This is done in the experiment in section 4.6. The quality of the estimated depth-maps is lower that the quality of the ground-truth depthmaps. It can be shown what the influence is of this quality drop of the depth-maps on the tracking accuracy of ORB-SLAM.

Therefore this experiment. ORB-SLAM has been ran with the Kitti images and corresponding ground-truth depthmaps according to the experimental setup described in section 4.1.1. All five the selected sequences from Kitti have been ran 20 times.

4.4.1. Results

The tracking accuracy of ORB-SLAM on the Kitti images together with the ground-truth depth-maps is shown in table 4.5. The mean μ and standard deviation σ of each error metric are presented. The results are presented next to the result of monocular ORB-SLAM from section 4.3 in figure 4.8. In this figure, the results of this experiment are added to figure 4.7.

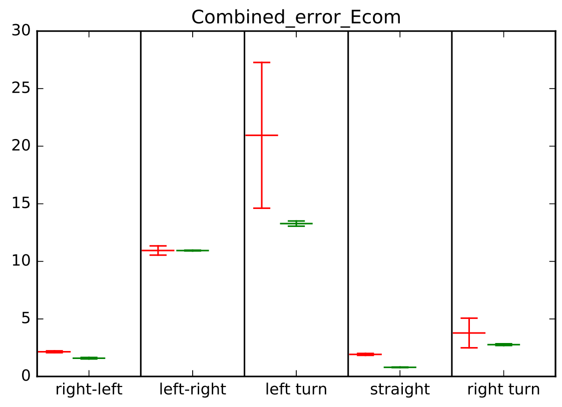
Again the ‘left turn’ and ‘left-right’ sequences result in relatively higher errors then the other sequences. Especially in the longitudinal error E_x, the yaw error E_a and in the positional error E_{pos}. Remarkable is the disappearance of the clear performance difference between the longitudinal error E_x and the lateral error E_y. With the added ground-truth depth information, the lateral error is quite comparable to the longitudinal error. On some sequences (for instance ‘right-left’) is the longitudinal error about twice the lateral error and on other sequences (for instance ‘left-right’) is the lateral error much higher than the longitudinal error. One is not always better then the other. Also the variance of these errors are much more comparable, also showing differences both ways dependent on the sequence.

The addition of the ground-truth depthmaps has mostly resulted in improvement in the tracking accuracy. The errors are smaller compared to the monocular version errors. Especially in the longitudinal error, E_x, and the positional error, E_{pos}, has the addition of ground-truth depthmaps resulted in much smaller errors. In the yaw error, E_a, is the difference much smaller but still there. In the lateral error, E_y, is this difference not clearly there as on some sequences the addition of ground-truth depth-maps resulted in higher lateral errors.

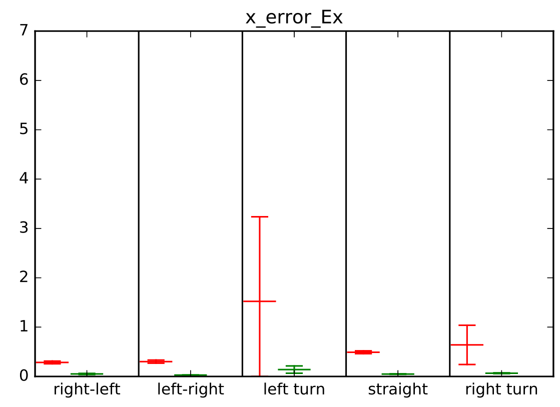
With these results, it has been proven that adding depth information to the inputs of ORB-SLAM does indeed improve the tracking accuracy. However, this setup is not monocular anymore and not applicable for the WEpod. Therefore, a CNN is going to be used to estimate the depth in the images. Before the CNN can be used to perform on Kitti images, it needs to be fine-tuned on Kitti images. This is the goal of the next experiment.

Error metric		**Sequence**				
		Right-Left	Left-Right	Left	Straight	Right
E_{com}	μ	1.586	10.936	13.279	0.794	2.764
	σ	0.148	0.076	0.514	0.055	0.171
E_x	μ	0.051	0.028	0.140	0.049	0.066
	σ	0.026	$4 \cdot 10^{-3}$	0.168	0.012	0.018
E_y	μ	0.025	0.073	0.029	0.014	0.054
	σ	$\cdot 10^{-3}$	$1 \cdot 10^{-3}$	$5 \cdot 10^{-3}$	$1 \cdot 10^{-3}$	$4 \cdot 10^{-3}$
E_a	μ	0.15	2.114	3.688	0.038	0.144
	σ	0.022	$2 \cdot 10^{-3}$	0.047	$3 \cdot 10^{-3}$	$9 \cdot 10^{-3}$
E_{pos}	μ	5.089	2.384	13.753	1.782	3.701
	σ	2.972	0.628	14.476	0.860	1.323

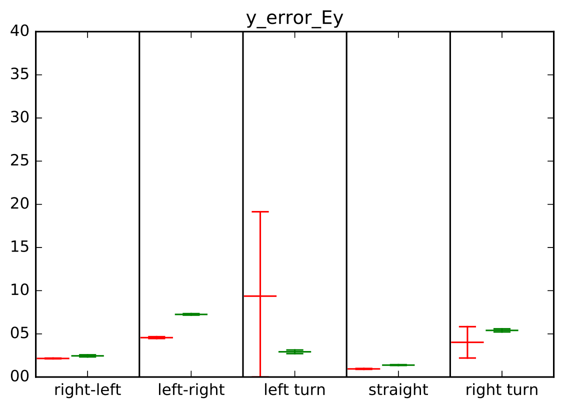
Table 4.5: The tracking accuracy of ORB-SLAM on the five different sequences from the Kitti image set together with the ground-truth depth-maps described in section 3.3. The error metrics and how there mean μ and standard deviation σ are described in section 4.1.1.



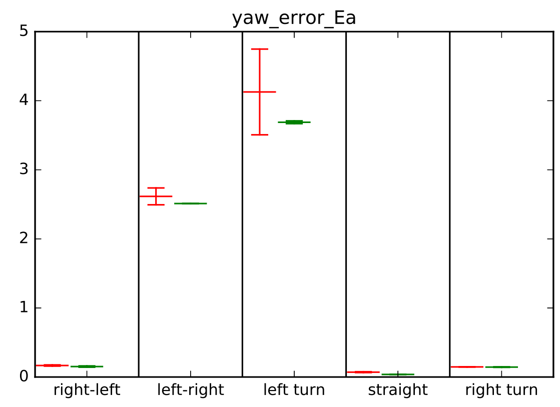
(a) The combined error E_{com}



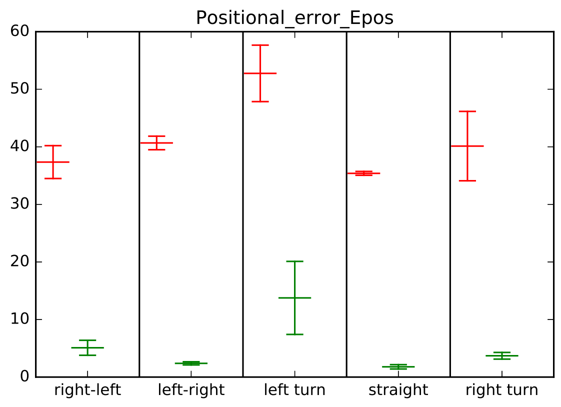
(b) The longitudinal error E_x



(c) The lateral error E_y



(d) The yaw error E_a



(e) The positional error E_{pos}

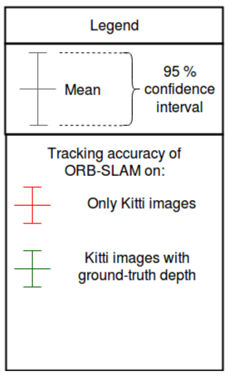

Legend
Mean
95 % confidence interval
Tracking accuracy of ORB-SLAM on:
Only Kitti images
Kitti images with ground-truth depth

Figure 4.8: The tracking accuracy of ORB-SLAM on only the Kitti images (red) and on Kitti images together with ground-truth depthmaps of these images (green) . The average value (middle, widest line) and 95 % certainty interval (upper and bottom smaller lines) of each metric is shown for each sequence.

4.5. Fine-tuning the CNN on KITTI images

The CNN used in this thesis is trained on the NYU dataset. This dataset consists of indoor scenery images and depth-maps. It needs to be fine-tuned on the Kitti dataset so it can estimate the depth in Kitti images.

This fine-tuning has been done using 7481 images and ground-truth depth-maps from different Kitti sequences. The ground-truth depth-maps have been created according to section 3.3 and the fine-tuning has done with the setup described in section 4.1.2. The parameters of the CNN have been initialized as their value in the original NYU trained version. The fine-tuning has been done with the hyper-parameters set as those found in section 4.2.

Only the training set of the images have been used to train on. The calculating of the performance has been done by letting the fine-tuned network estimate the depth in the testing image set. These are images it has never seen before, to rule out overfitting. The estimated depth-maps of these testing images have been compared to the ground-truth depth-maps according to the metrics described in section 4.1.2.

4.5.1. Results

The performance of the depth-estimating CNN fine-tuned on Kitti images can be seen in table 4.6. The average values of each metric over all the images in the testing set have been presented. Also the performance of the original network is shown. These values have been obtained from the original paper [31]. The squared relative difference metric was not reported in this paper.

The network fine-tuned on Kitti images performs comparable to the original network from the paper, see table 4.6. The log RMSE and absolute relative difference metrics are about the same. The network fine-tuned on Kitti images only performs ever so slightly worse then the network on NYU images. On the RMSE metric, a big difference was noticeable. This difference is caused since the RMSE metric is not relative to the scale of the depth estimated. The NYU images are of indoor scenarios where a typical depth value is about 5 meters, while a typical depth value of Kitti images is 30 meters. A typical depth value in the Kitti images is about 6 times as big as in the NYU images. This scale difference is taken into account by dividing the RMSE of the network fine-tuned on Kitti images by 6. The result is comparable to the RMSE of the network trained on NYU images. Table 4.6 shows the scaled RMSE value and between brackets the original value.

Examples of the resulting depth-maps are provided. In figure 4.9, the estimated depth-maps by the CNN before fine-tuning and after fine-tuning are shown to state the difference fine-tuning makes. In figure 4.10, three more examples of the estimated depth-maps are shown. These are three images of different road scenarios from the testing set of images. These images have not been used to train on.

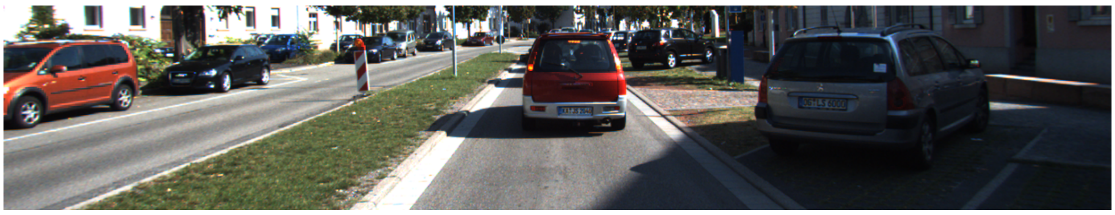
The image is figure 4.9 show that the fine-tuning is necessary and improves the result. The estimated depth-map from before fine-tuning shows 'ghostly' objects, not-existent patterns and it misses the overall ground-plane. This clear ground-plane is visible in estimated depth image from the fine-tuned network. In this images, the different cars, poles and signs are visible. This shows that the network has detected these obstacles. Also no more 'ghostly' objects that are not present in the original image are visible. The maximum depth in this image is 50 m.

Figure 4.10 shows that the network performs well for different types of images. The left image consists of a scene with not a lot of obstacles. The middle image consists of a road with more obstacle on the side: differently orientated cars, trees and houses. The right image shows a car crossing an intersection in front of the camera. On all these different images, the network is able to recognize the different objects as being closer and correctly estimates which object is closer by.

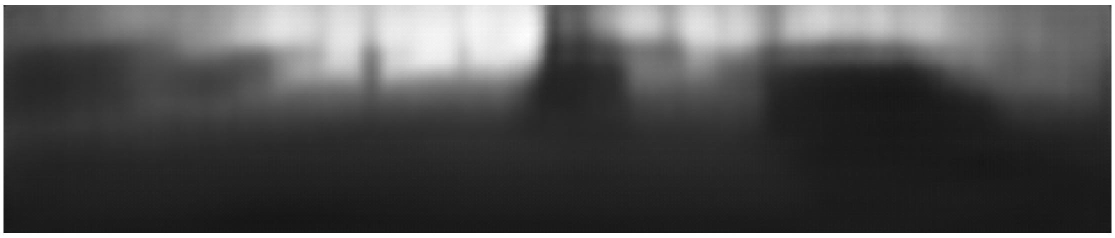
It is possible to do ORB-SLAM on image with depth information from the CNN fine-tuned on Kitti images while keeping the whole setup monocular. The depth-estimating CNN can be added to ORB-SLAM and the resulting tracking accuracy can be compared to that of monocular ORB-SLAM and ORB-SLAM with ground-truth depth-maps. This is done in the next experiment.

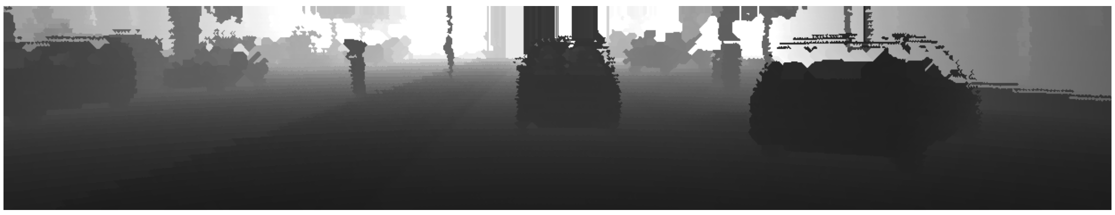
	RMSE [m]	log RMSE [m]	abs rel diff [m]	squared rel diff [m]
Original network [31]	0.573	0.195	0.127	-
Kitti fine-tuned network	0.661 (3.965)*	0.219	0.167	1.054

Table 4.6: The performance of the depth-estimating CNN fine-tuned on Kitti images. The network estimated the depth in the testing set of images, which it has never seen before, and the results are compared with the ground-truth through the metrics described in section 4.1.2. The average of each of these metrics is presented together with the results from the original authors which trained it on the NYU dataset [31]. *The presented value is divided by 6 to account for the scene scale difference. The original value is presented between the brackets.


(a)

(b)


(c)


(d)

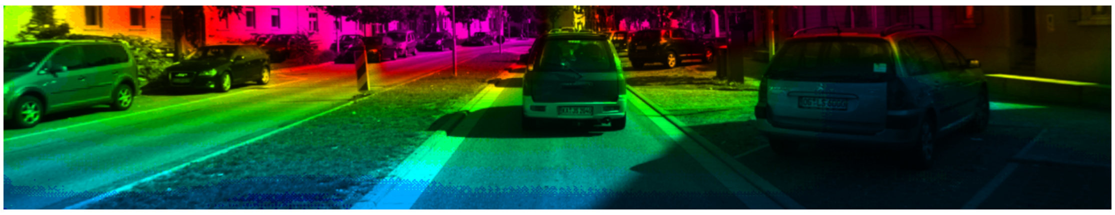

(e)

Figure 4.9: a: An example input image from the Kitti testing group. b: the depth image estimated by the CNN without fine-tuning, so only trained on the NYU dataset. c: the depth image estimated by the CNN fine-tuned on the Kitti training set. d: the ground truth depth image corresponding to this image. e: The input image with the depth as the color (blue to purple = close to far).

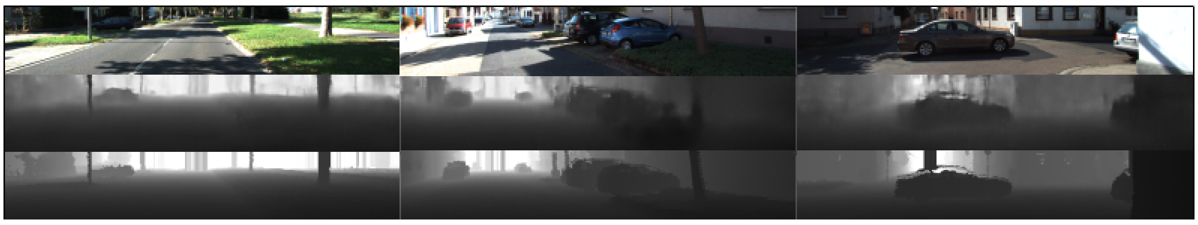


Figure 4.10: Top: Three different input images from the testing Kitti image set. Middle: The resulting output depth image from the CNN fine-tuned on the Kitti training set. Bottom: The groundtruth depth image.

4.6. ORB-SLAM on KITTI images with CNN estimated depthmaps

The CNN fine-tuned on the Kitti images can now be combined with ORB-SLAM. The images are passed into the depth estimating CNN and into ORB-SLAM together with the estimated depth-map. ORB-SLAM does calculate the trajectory based on images and their depth-map but the whole setup is still monocular. This experimental setup is the same as described in section 4.1.1. Again all the 5 selected sequences from the Kitti dataset are ran 20 times each.

4.6.1. Results

The tracking accuracy of ORB-SLAM on Kitti images with the estimated depth-maps is shown in table 4.7. The mean μ and standard deviation σ of each error metric on each sequence are presented. In figure 4.11 the mean and 95 % certainty interval is presented for each error metric and each sequences. The tracking accuracy of ORB-SLAM on Kitti images with the estimated depth-maps is presented together with the tracking accuracy of ORB-SLAM on Kitti images with ground-truth depth-maps, and the tracking accuracy of the monocular ORB-SLAM.

The ‘left turn’ and ‘left-right’ appear to again be the ones where the errors are substantial higher. The ‘right-left’ sequence this time also shows higher errors, especially on the positional orientated error metrics. On the longitudinal error E_x, the lateral error E_y and the positional error E_{pos} is the performance on the ‘right-left’ sequence worse then on the other sequences.

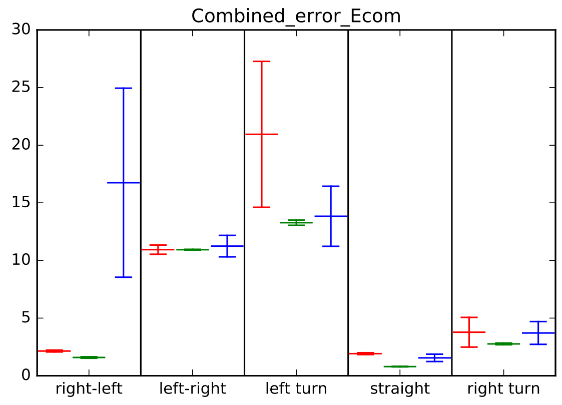
The longitudinal error E_x and the lateral error E_y show a clear difference. On each sequence is the mean of the longitudinal error higher then the mean of the lateral error. Also is the variance of the longitudinal error is higher then the variance of the lateral error.

The performance on these metrics of ORB-SLAM with the depth estimating CNN falls in between the performance of monocular ORB-SLAM and the performance of ORB-SLAM with ground-truth depth-maps. On a few sequences some metrics perform better then ORB-SLAM with ground-truth depth-maps. An example is the yaw error E_a on the ‘left turn’ sequence. Here ORB-SLAM with the CNN outperforms ORB-SLAM with the ground-truth depth-maps. ORB-SLAM with the CNN sometimes also performs worse than the monocular version. For instance with the longitudinal error E_x on the ‘left-right’ sequence. The variance of ORB-SLAM with the CNN usually lays in between the variance of monocular ORB-SLAM and ORB-SLAM with ground-truth depth-maps.

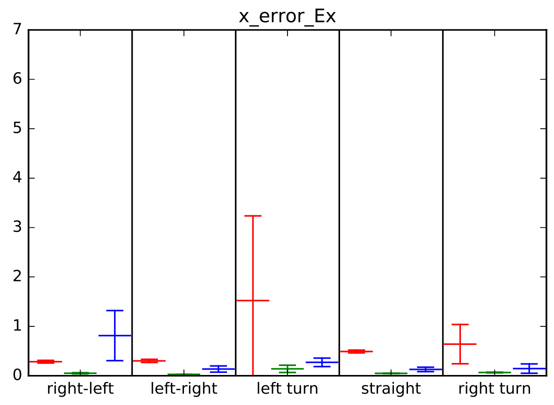
This experiment has shown that over all the addition of the depth estimating CNN to monocular ORB-SLAM does improve the tracking accuracy on the Kitti images. The question remains if this is also the case on WEpod images. To be able to test this out, a depth estimating CNN that is capable of estimating the depth in WEpod images is necessary. In the following experiment, it is investigated how the depth-estimating CNN fine-tuned on Kitti images performs on WEpod images.

Error metric		**Sequence**				
		Right-Left	Left-Right	Left	Straight	Right
E_{com}	μ	16.748	11.246	13.835	1.552	3.712
	σ	18.711	2.125	5.944	0.726	2.254
E_x	μ	0.813	0.136	0.272	0.129	0.145
	σ	1.157	0.143	0.196	0.102	0.215
E_y	μ	0.319	0.082	0.100	0.024	0.070
	σ	0.467	0.024	0.094	0.014	0.055
E_a	μ	0.629	2.405	2.880	0.087	0.173
	σ	0.422	0.581	1.452	0.115	0.105
E_{pos}	μ	16.402	8.524	9.975	5.355	5.724
	σ	14.145	5.388	8.815	2.980	2.894

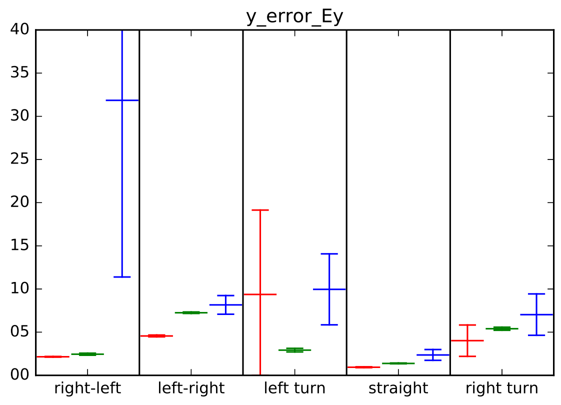
Table 4.7: The tracking accuracy of ORB-SLAM on the five different sequences from the Kitti image set together with the estimated depth-maps by the CNN from the experiment in section 4.5. The error metrics and how there mean μ and standard deviation σ are described in section 4.1.1.



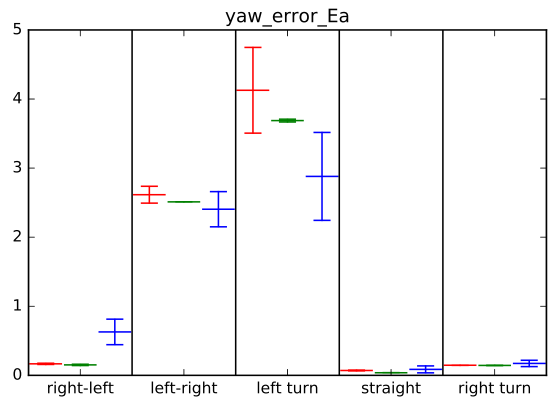
(a) The combined error E_{com}



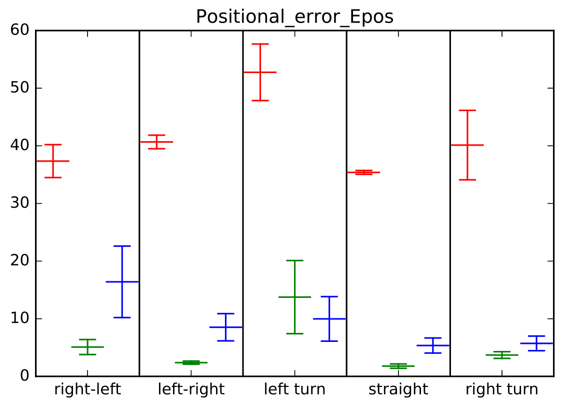
(b) The longitudinal error E_x



(c) The lateral error E_y



(d) The yaw error E_a



(e) The positional error E_{pos}

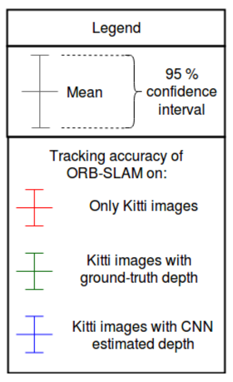

Legend
Mean
95 % confidence interval
Tracking accuracy of ORB-SLAM on:
Only Kitti images
Kitti images with ground-truth depth
Kitti images with CNN estimated depth

Figure 4.11: The tracking accuracy of ORB-SLAM on only the Kitti images (red), on Kitti images together with the CNN estimated depth-maps (blue) and on Kitti images together with ground-truth depth-maps of these images (green) . The average value (middle, widest line) and 95 % certainty interval (upper and bottom smaller lines) of each metric is shown for each sequence.

4.7. Predicting depth in WEpod images with the KITTI fine-tuned CNN

A CNN that predicts the depth in WEpod images is needed to add it to ORB-SLAM. This means that the depth estimating CNN needs to be able to estimate depth-maps of the WEpod images. It is possible that the CNN fine-tuned on the Kitti images can be used to estimate the depth in WEpod images. This is possible because both the Kitti images and the WEpod images are both of outdoor road scenery. Estimating the depth in WEpod images could save a lot of effort fine-tuning the depth estimating CNN on WEpod images. This could indicate that one depth estimating CNN version can be used on similar images from different datasets.

The WEpod images however do differ from the images from the Kitti dataset. They have been recorded with different camera's, which have different intrinsic parameters and distortion effects. The images all have been undistorted, but this can never be done perfectly. There still exists some slight distortion differences. The images also have different aspect ratio's and sizes. Even though the images are both from outdoor road scenarios, they have been recorded on different roads with different surroundings in different countries.

The CNN fine-tuned on Kitti images has been used to predict the depth in images from the WEpod, at their original size and aspect ratio. As shown in figure 4.12 and as expected, the network is not able to create realistic depth-maps for these images.

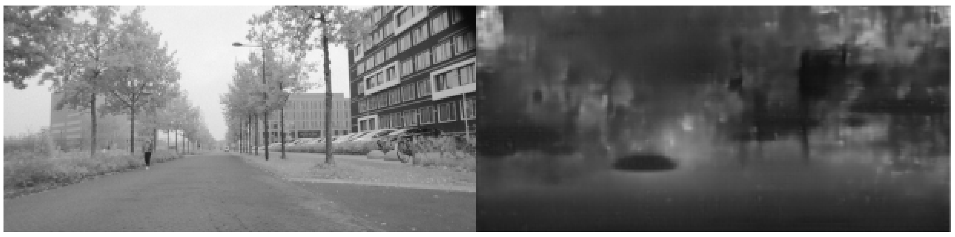


Figure 4.12: Left: The input WEpod image. Right: The resulting output depth image form the CNN trained with L2 loss, trained for 25 epochs with lr0.0001epd1dr0.95.

Multiple things are changed to see how the performance of this network can be improved. First of all are the WEpod images cropped to the aspect ratio of Kitti images and then resized to the same size. The images have been cropped such that the horizon stays in the center of the images. This improves the resulting depth-maps. The resulting depth-maps after only cropping to the aspect ratio of Kitti images can be seen in figure 4.13. The resulting depth-maps after cropping and then resizing to the same size can be seen in figure 4.14. Next the images have been cropped and resized such that the horizon is placed at the same height as it is in the Kitti images. This really improved the resulting depth-maps, see figure 4.15.

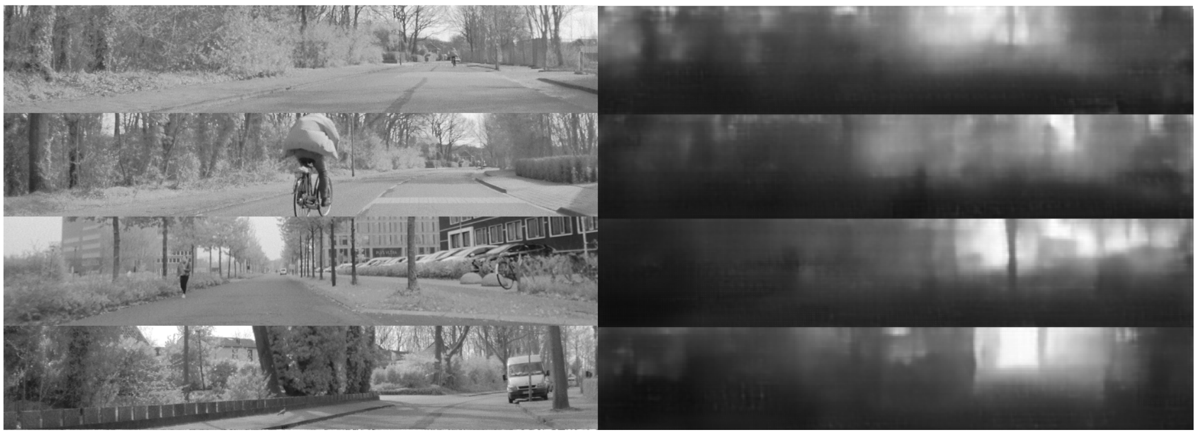


Figure 4.13: Left: The WEpod images cropped to the same aspect ratio as the Kitti images with the horizon in the middle. Right: The resulting estimated depth image.

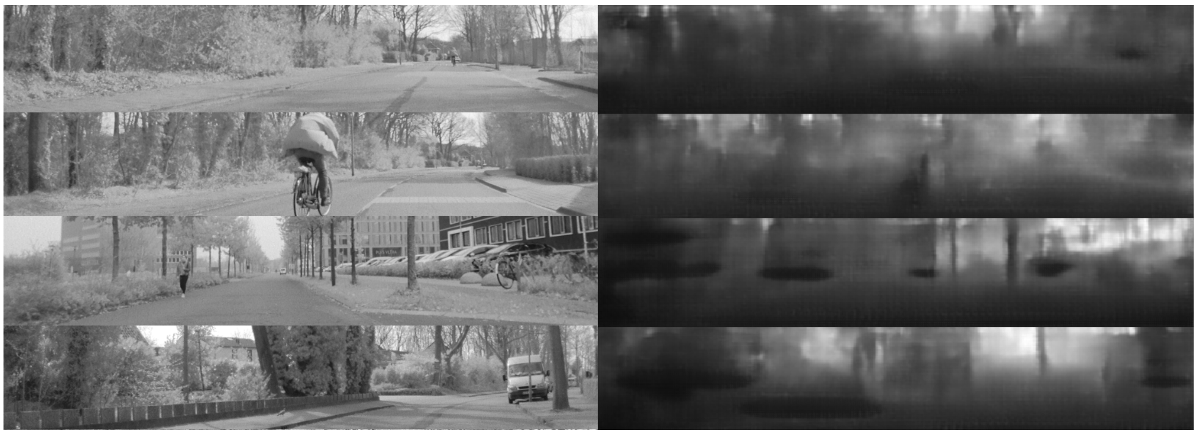


Figure 4.14: Left: The WEpod images cropped to the same aspect ratio as the Kitti images with the horizon in the middle and resized to the same size. Right: The resulting estimated depth image.

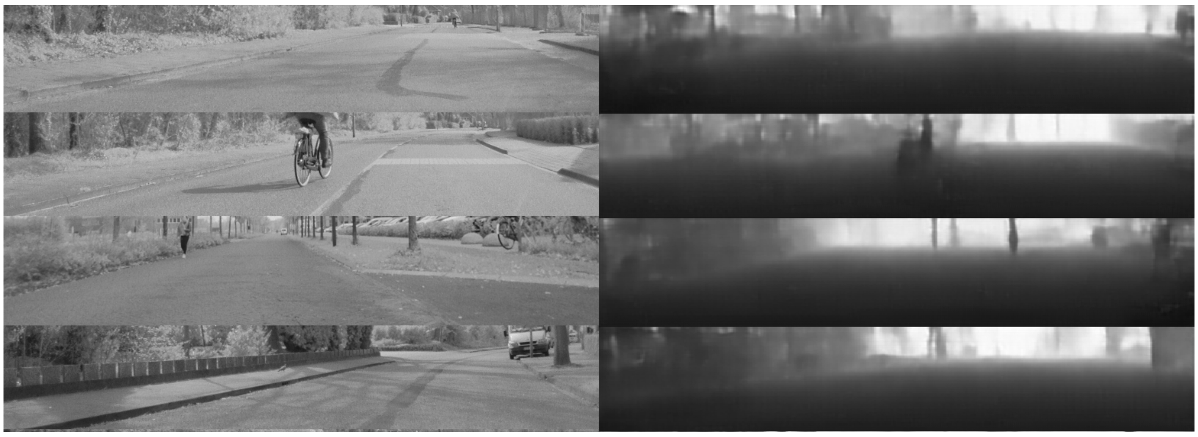


Figure 4.15: Left: The Wepod images cropped to the same aspect ratio, with the horizon at the same height and resized to the same size as the Kitti images. Right: The resulting estimated depth image.

The network fine-tuned on the Kitti dataset is used to predict 5 different sets of WEpod images. Each set consists of the same 642 WEpod images, cropped and resized in different ways. From these images, also ground-truth depth-maps are created using the S21 stereo setup as described in section 3.3.

The first set of images consists of the images cropped such that the horizon is about in the same height as in the Kitti images. For the second set of images, the images are corrected for the pitch angle of the camera. The horizon now stays at the same height, even when going over a speed bump. The third set of images are corrected for the pitch and roll angles of the camera. In this set the horizon is always on the same height and even always at the same angle as in the Kitti dataset, even when while taking a corner the vehicle slightly leans to one side. Two more sets are created by applying a Gaussian filter on the images, filtering out noise from the camera. This creates a total of five different sets. The performance of the depth-estimating network on these six sets can be compared to choose one way of preparing the WEpod images for the depth-estimating CNN.

The performance of the depth-estimating CNN fine-tuned on Kitti images and predicting the depth in the five different types of WEpod images is shown in figure 4.8. In this table, all four the metrics defined in section 4.1.2 are presented for each instance.

The cropping of the images based on the current pitch value does not increase the performance of the network. For a select amount of images, those for instance taken when going over a speed bump, does the cropping based on the pitch value increase the performance of the network. However, the performance on the other images apparently worsens. The rotating of the images based on the roll value worsens the performance of the network and the Gaussianing of the images to remove the noise does not appear to have any big effect.

The performance of the depth-estimating CNN fine-tuned on Kitti images and predicting on WEpod images has been investigated for different ways of preparing WEpod images to the size of Kitti images. This CNN could be added to monocular ORB-SLAM on WEpod images. It is also possible to fine-tune the network on WEpod images. The performance of the network fine-tuned on WEpod images is investigated in the next experiment.

Fine-tuned on	Predicted on	RMSE	log RMSE	abs rel diff	squared rel diff
KITTI	KITTI	3.983	0.216	0.163	1.034
KITTI	WEpod	7.084	0.336	0.264	2.282
KITTI	WEpod C	6.978	0.386	0.391	2.930
KITTI	WEpod RC	8.245	0.433	0.442	4.634
KITTI	WEpod GC	6.979	0.386	0.392	2.934
KITTI	WEpod GRC	7.155	0.390	0.389	2.937

Table 4.8: Resulting metrics KITTI fine-tuned, predicted on KITTI and on WEpod. G = Gaussianed with 5x5 and sigma 1, R = rotated to compensate the roll angle, C = cropped to compensate for the pitch angle

4.8. Fine-tuning the CNN on WEpod images

Another way of creating a CNN that can estimate the depth in WEpod images is by fine-tuning the network directly on WEpod images. The ground-truth depth-maps of WEpod images can be created with the S21 stereo setup as described in section 3.3. The network can directly be fine-tuned on images from the WEpod, the same dataset of images it needs to predict on. This CNN fine-tuned on WEpod can also be combined with ORB-SLAM as in figure 1.2.

The network is fine-tuned on the training set of the 642 WEpod images together with the corresponding ground-truth depth-maps. The fine-tuning has been done using the same hyper-parameters as found in section 4.2 and the parameters have been initialized as the value's that they have after fine-tuning on the Kitti dataset. The performance of the CNN fine-tuned on WEpod images is evaluated with the testing set of the 642 WEpod images. All according to the setup described in section 4.1.2.

4.8.1. Results

The resulting performance of the depth-estimating CNN fine-tuned on WEpod images is shown in table 4.9. The CNN is used to predict the WEpod images from the testing set and the resulting depth-maps are compared to the ground-truth depth-maps. The average value of each metric over all the estimated WEpod depth-maps in the testing set is reported in the table. Also the best results from the experiments described in sections 4.5 and 4.7 are shown for easy comparison.

It can be seen in table 4.9 that the performance of the WEpod fine-tuned network is not better. It is even slightly worse then the performance of the network fine-tuned on the Kitti images on WEpod images. The performance of the Kitti fine-tuned network on Kitti images is about twice as good as the performance of the WEpod fine-tuned network on WEpod images. Further investigation has been done into the resulting depth images. One example can be seen in figure 4.16.

From figure 4.16 it is clear that the network is not performing well. It misses all detail in the image, abstracting the depth up until no obstacles are recognizable in the image. Only the basic pattern of the depth-map getting deeper and deeper toward the middle-top of the image is present in the estimated depth image. Each estimated depth image from the network is about the same. The network has learned to create a basic abstract depth-map without any detail as a solution. Also is the road directly in front of the WEpod not covered by the ground-truth depthmaps. The network thus did not learn to estimate the depth here correctly. These detail-less solutions do not perform terrible: only slightly worse than the Kitti fine-tuned network on WEpod images and only about twice as worse as the Kitti fine-tuned network on Kitti images (table 4.9).

Two networks have been created that estimate the depth in WEpod images, one with many details and one without any details. These can be combined separately with ORB-SLAM. The tracking accuracy of both combinations can be compared to each other to see if this detail is necessary. This is done in the next experiment.

Fine-tuned on	Predicted on	RMSE	log RMSE	abs rel diff	squared rel diff
KITTI	KITTI	3.983	0.216	0.163	1.034
KITTI	WEpod	7.084	0.336	0.264	2.282
WEpod	WEpod	7.698	0.401	0.371	3.700

Table 4.9: Resulting metrics KITTI fine-tuned, predicted on KITTI and on WEpod. C = cropped to pitch.

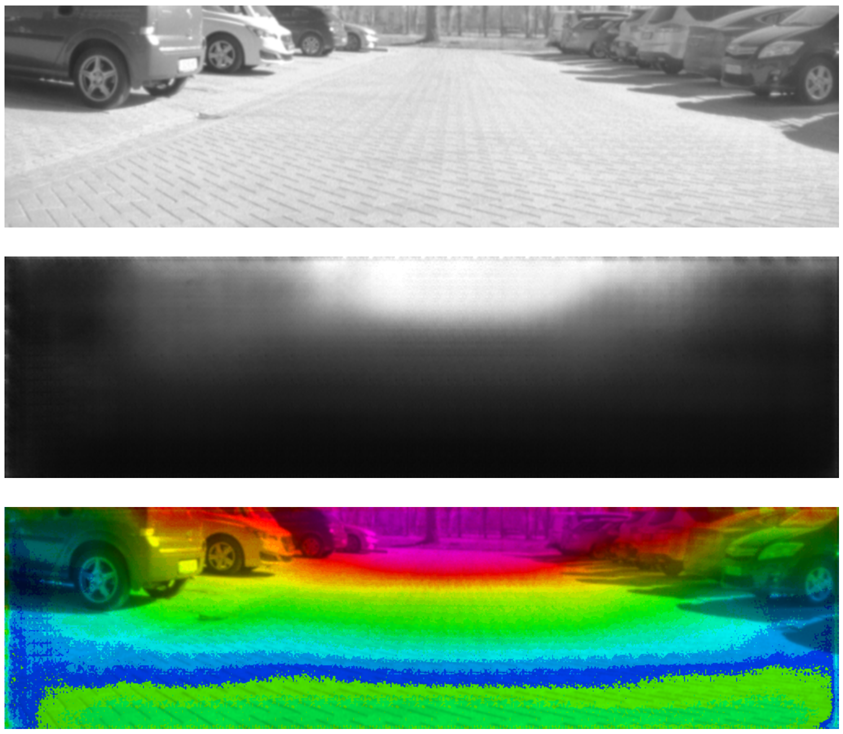


Figure 4.16: Top: An example input image from the WEpod testing group. Middle: the depth image estimated by the CNN fine-tuned on the WEpod images. Bottom: The input image with the log of the depth value determining the color, ranging from blue (close) to purple (far).

4.9. ORB-SLAM on WEpod images with CNN estimated depthmaps

There are two versions of the CNN that estimate the depth in WEpod images. Both have been combined with ORB-SLAM and the tracking accuracy can be compared. ORB-SLAM estimates the trajectory of WEpod images with estimated depth-maps. The whole system (ORB-SLAM + CNN) is still monocular, it only takes the images as inputs. ORB-SLAM together with the CNN's in this experiment are setup as described in section 4.1.1. This experiment has been ran on the five selected sequences from the WEpod data, 20 times per sequence.

4.9.1. Results

The tracking accuracy of ORB-SLAM with estimated depth-maps from the CNN fine-tuned on Kitti images is shown in table 4.10. The tracking accuracy of ORB-SLAM with estimated depth-maps from the CNN fine-tuned on WEpod images is shown in table 4.11. The mean μ and standard deviation σ of each error on each sequence are presented. The difference between both versions is visible in figure 4.17.

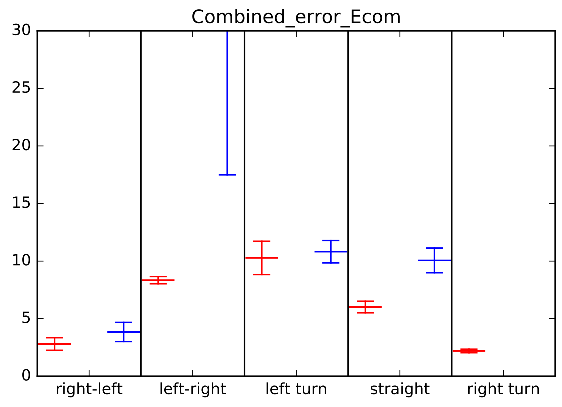
ORB-SLAM with the network fine-tuned on Kitti images outperforms ORB-SLAM with the network fine-tuned on WEpod images. On almost all the metric and sequence combinations does the Kitti fine-tuned version achieve lower errors and variances. Only in the positional error metric E_{pos} with the 'right turn' and 'left turn' sequence and in the lateral error metric E_y with the 'straight pass' metric does the WEpod fine-tuned version achieve lower errors then the Kitti fine-tuned version.

Error metric		**Sequence**				
		right	left	straight pass	straight	right-left
E_{com}	μ	2.798	8.346	10.276	6.012	2.194
	σ	1.251	0.715	3.298	1.142	0.345
E_x	μ	0.146	0.547	0.987	0.476	0.123
	σ	0.067	0.045	0.343	0.085	0.016
E_y	μ	0.041	0.112	0.148	0.088	0.037
	σ	0.028	0.017	0.079	0.027	$9 \cdot 10^{-3}$
E_a	μ	0.242	0.766	0.563	0.393	0.131
	σ	0.181	0.079	0.197	0.104	0.015
E_{pos}	μ	15.395	23.252	29.657	15.102	8.567
	σ	4.914	4.082	10.188	3.186	3.418

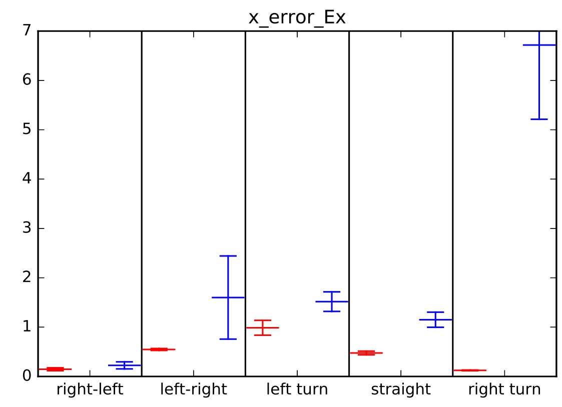
Table 4.10: The tracking accuracy of ORB-SLAM on the five different sequences from the WEpod image set together with the estimated depth-maps by the CNN fine-tuned on Kitti images from the experiment in section 4.5. The error metrics and how there mean μ and standard deviation σ are described in section 4.1.1.

Error metric		**Sequence**				
		right	left	straight pass	straight	right-left
E_{com}	μ	3.843	33.339	10.817	10.06	46.134
	σ	1.897	36.154	2.215	2.443	19.869
E_x	μ	0.225	1.600	1.517	1.150	6.717
	σ	0.163	1.923	0.453	0.350	3.434
E_y	μ	0.061	0.344	0.101	0.107	0.500
	σ	0.045	0.568	0.044	0.049	0.430
E_a	μ	0.265	4.860	0.862	0.864	2.607
	σ	0.177	8.764	0.191	0.371	1.476
E_{pos}	μ	7.952	17.068	37.708	25.363	32.181
	σ	1.716	7.836	10.505	9.175	20.550

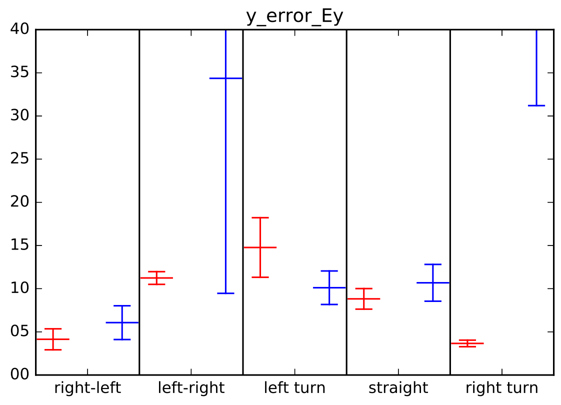
Table 4.11: The tracking accuracy of ORB-SLAM on the five different sequences from the WEpod image set together with the estimated depth-maps by the CNN fine-tuned on WEpod images from the experiment in section 4.8. The error metrics and how there mean μ and standard deviation σ are described in section 4.1.1.



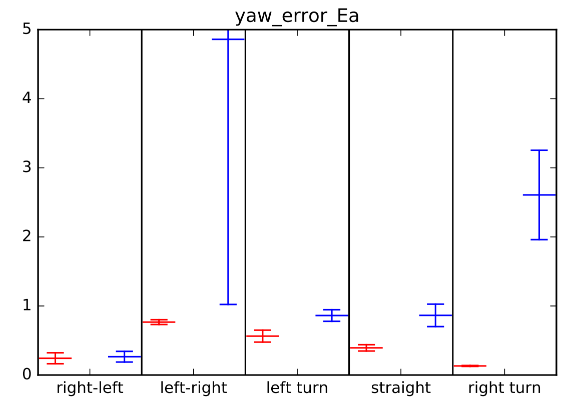
(a) The combined error E_{com}



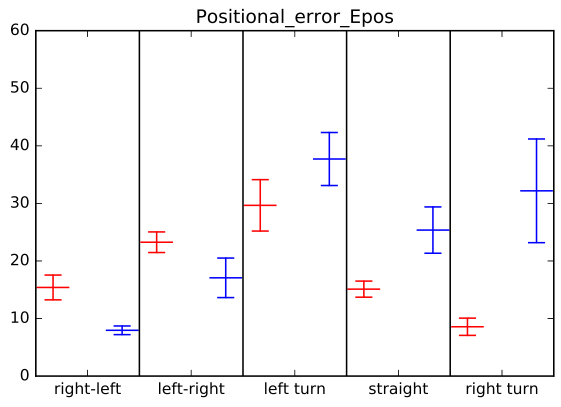
(b) The longitudinal error E_x



(c) The lateral error E_y



(d) The yaw error E_a



(e) The positional error E_{pos}

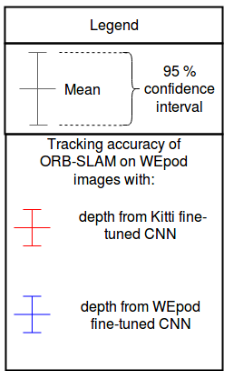

Legend
Mean
95 % confidence interval
Tracking accuracy of ORB-SLAM on WEpod images with:
depth from Kitti fine-tuned CNN
depth from WEpod fine-tuned CNN

Figure 4.17: The tracking accuracy of ORB-SLAM together with the depth estimating CNN fine-tuned on Kitti (red) or WEpod (blue) images, with the five WEpod sequences as inputs. The average value and 95 % certainty interval of each metric is shown for each sequence.

5 Discussion & Conclusion

5.1. Discussion

The WEpod is a self-driving vehicle that needs to improve its localizing capabilities. It localizes using GPS and lidar information. This is not robust and accurate enough to safely navigate through the traffic. The monocular vision-based SLAM algorithm ORB-SLAM is investigated as a localizing method addition to the existing methods in the WEpod. Monocular ORB-SLAM on WEpod images failed to initialize and thus to perform SLAM. Therefore, the goal of the experiments was to investigate if the addition of a depth estimating Convolution Neural Network (CNN) to monocular ORB-SLAM on WEpod images improves its tracking accuracy.

ORB-SLAM has a special way of initializing compared to other state of the art monocular SLAM methods. ORB-SLAM tries to initialize the surrounding scene with the fundamental matrix and the homography. They give both initialization methods a score and if one of the scores is high enough, initialization is done. If no initialization method scores high enough, initialization is tried again with a new frame. Monocular ORB-SLAM on different sequences of the WEpod images (5 sequences, 20x each) failed to initialize throughout 99 from the 100 runs. The purpose of this thesis is not to investigate why monocular ORB-SLAM fails on WEpod images, but how to get ORB-SLAM working on WEpod images.

The fact that the state of the art monocular SLAM methods do not initialize as robustly as desirable has been acknowledged by the scientific community. The authors of the monocular SLAM methods came forth with upgraded versions. They improved the initialization and results of their SLAM methods by added depth-information to the inputs. This depth information can be obtained by using other methods.

The WEpod does not have sensors that make it possible to create dense and accurate depth-maps of its images and these cannot be added. This means that another way of obtaining depth-maps of the images is necessary. This thesis research investigates the possibility of using a depth-estimating CNN to estimate the necessary depth information.

Adding a depth-estimating CNN to ORB-SLAM has been investigated first with images from the Kitti dataset. This was done as monocular ORB-SLAM initializes on the Kitti dataset. It makes it possible to better compare the tracking accuracy of monocular ORB-SLAM with the methods described in this thesis. Monocular ORB-SLAM on Kitti images acts as the baseline for the other methods on the Kitti dataset.

It is noticeable that monocular ORB-SLAM has more trouble with the cornering sequences then the straight sequence, especially in the variance of its results. It has been observed that ORB-SLAM tends to have difficulty with rotational movements. It would appear that ORB-SLAM has much more trouble with the sequences with left turns then the right turns. There are multiple possible causes: First of all, it could be a coincidence as the algorithm does not treat them different. It is possible that it coincidental was harder to find and match features in the sequences with the left turns. Secondly, a right turn is sharper then a left turn as the car was driving on the right side of the road. More different sequences with left and right turns would need to be examined to be able to conclude anything on these possible causes.

The Kitti dataset is chosen as 3D lidar points from a 64-plane lidar are included. These 3D points have been used to create ground-truth depth-maps. These depth-maps make it possible to not only evaluate the performance of the depth-estimating CNN on Kitti images with, but also to perform ORB-SLAM on Kitti images with ground-truth depth-maps. The tracking accuracy of ORB-SLAM on Kitti images with ground-truth depth-maps has been compared to the tracking accuracy of monocular ORB-SLAM.

On almost every sequence and for every error does ORB-SLAM on images with ground-truth depth-maps achieve a lower average error and variance. Adding the ground-truth depthmaps seem to especially help with the estimation of the longitudinal movement. The average error in the lateral and angular directions does not appear to improve much. This could be because there is very little angular and lateral movement in comparison to the longitudinal movement in the sequences. Also, it is known that monocular vision-based SLAM performs relatively good for lateral movements. There is less room for improvement. The variances of the errors have consistently improved. It can be concluded that adding ground-truth depth-maps to the inputs of ORB-SLAM does increase the tracking accuracy.

The next step is evaluating the tracking accuracy of ORB-SLAM together with a depth-estimating CNN on Kitti images. A well-performing depth-estimating CNN that has been proven to work well in cooperation with SLAM has been chosen. This network has been trained on images of indoor scenarios, thus it needs to be fine-tuned on the Kitti images. The hyper-parameters used for the fine-tuning of the network have been chosen empirically by comparing the results of networks trained differently. This does not ensure that this set is the absolute best, but it does give enough reason to assume that with this set, the network trains adequately.

The network has been fine-tuned on the Kitti images and gives adequate results on different types of Kitti images. The results on these images are comparable to the results that the authors mentioned on the original dataset. There are some aspects of the ground-truth depth-maps that the CNN has inherited from the ground-truth data. The ground-truth depth-maps have been created using lidar points. This results in certain parts of the ground-truth depth-maps where the depth is not correctly known, such as the spaces in between the points. This problem could partly be solved by linear interpolating the points. However, other problems still exist in the ground-truth depth-maps.

One of these problems is that the windows of cars are missed. The lidars 'looks through' the windows and senses the depth behind the windows. The shiny polish of the cars also reflect the lidar beams. The network however does not learn these faults in the ground-truth depth-maps, cars are actually detected very well.

A fault in the ground-truth depth-maps that the network does learn is the falsely stretched out objects in the top-middle of the images. The lidars have a limited range in which they can sense depth. This results in the top middle patch of the images being uncovered. The lidar beams pointing in this direction go off into the horizon without returning. While creating the ground-truth depth-maps, these unknown pixels have been filled up by copying the depth from the highest known point beneath the unknown pixel. This has created stretched out objects in the top-middle of the images, which have been learned by the depth-estimating CNN. However, it is not assumed to be a big problem because this only effects a very small portion of the image.

The CNN fine-tuned on Kitti images can now be combined to ORB-SLAM on Kitti images. This setup is still monocular and benefits from the advantages of depth information. The performance in the tracking accuracy metrics are usually in between the performance of monocular ORB-SLAM and ORB-SLAM with the ground-truth depth-maps. In some cases it performed worse than the monocular ORB-SLAM and also some cases where it performed better then ORB-SLAM with ground-truth depth-maps. This kind of slight inconsistency lays inherently in the way ORB-SLAM works. ORB-SLAM is heavily based on optimization algorithms, which can result in very different outcomes with even a slight variation in the starting values. From these results it can be concluded that adding a depth-estimating CNN to monocular ORB-SLAM on Kitti images does indeed improve the tracking accuracy.

With this knowledge, the ORB-SLAM with depth estimating CNN setup can be tested on WEpod images. The depth-estimating CNN for this network can be created in multiple ways. Two of these versions have been investigated in this research as these two are the most obvious. One network that has been investigated is the network fine-tuned on Kitti images. This network would need to perform on WEpod images.

The usage of a CNN cross-dataset, fine-tuning on one dataset and predicting on another, is not guaranteed to work. The images from different datasets describe different scenarios, have been taken with different camera and have been processed differently. The images from the Kitti and WEpod datasets are assumed to be very similar. Both are from road scenarios from north-west Europe and both have been undistorted. Investigation into how the WEpod images can be prepared for the Kitti fine-tuned network has been done. They could need to be resized to the Kitti size and can be passed through Gaussian filter to remove noise.

The WEpod images need to be cropped and resized to the same aspect ratio and size as the Kitti images. The network has only seen images from this size and aspect ratio and thus has not learned to be robust against differently sized images. With this cropping, the horizontal field of view of the images has not taken into account. As many horizontal information as possible has been kept in the WEpod images by taking the total width and adjusting the height cropped such that the aspect ratio is correct. From the experiments, it is clearly shown that this process of cropping and re-sizing to the right aspect ratio and size increases the quality of the estimated depth-maps.

The height of the horizon in the images matters a lot. The images where first cropped such that the horizon was in the middle. The network did not perform well on these images. When the images where cropped such that the horizon was at the same height as in the Kitti images, the result improved. This gives reason to believe that the depth-estimating CNN could learn a transformation between the images and the 3D-world, especially for the ground-plane. This ground-plane is something constant in each image and could be a basic starting point for the CNN.

Based on the previous findings, more investigation into the position of the horizon has been done. It was noticeable that that the performance of the network would drop when the WEpod would drive over a speed-bump as the horizon would shift. The WEpod records its roll and pitch angle with a IMU, thus it was possible to account for this. The cropping of the WEpod images has been done based on the pitch angle and the image has been rotated based on the roll angle. This way the horizon would always be on the same place in the images.

The cropping of the image based on the pitch angle hardly made any difference. Even though the estimated depth of the images affected by speed-bumps did appear to improve, the impact overall was not noticeable in the metrics. This could be because the results on the other images became slightly worse. The rotating of the image to account for the roll angle did not improve the result, it even worsened the result. This could be because there would be small sections of the image that would be black due to the rotating of the image or because the roll angle values from the WEpod are not accurate enough.

The WEpod images have also been passed through a Gaussian filter before being fed into the depth-estimating CNN. This Gaussian filter filtered out the noise in the WEpod images, which are originally quite noisy. Surprisingly, this did not appear to have an affect on the performance of the network. This shows that the network does not concentrate on the small details, but more on the greater picture. It probably tries to find larger objects such as cars, trees, poles and houses. Then it estimates the depth of these larger objects and adds them to a basic ground-plane depth-map. It could do some kind of averaging, and thus filtering out the noise inherently in its process.

The performance of the depth-estimating CNN fine-tuned on Kitti is the best on the WEpod images that have been cropped to the same aspect-ratio and re-sized to the size of Kitti images without accounting for pitch and roll. The performance on WEpod images is worse than on Kitti images, but this makes sense. This performance has been compared to the other option: fine-tuning the depth-estimating CNN on WEpod images.

Obtaining the necessary data for the fine-tuning on WEpod images came with some challenges as the vehicle does not have suitable sensors. The WEpod has temporary been outfitted with a S21 Multisense stereo setup to obtain depth information that could be projected to the images. This is far from a practical solution to use consistently on the WEpod, but it did serve its purpose to obtain a dataset. The set of WEpod images with depth-maps that was possible to obtain during the period of this thesis research consists of 642 images, much less then the 7481 images used from the Kitti dataset. The images and depth-maps from the WEpod are all from one sequence in one area.

When the depth-estimating CNN is fine-tuned on these WEpod images something interesting happens. The performance according to the metrics is certainly not bad. It performs slightly worse the the Kitti fine-tuned version but not horrible. However, the depth images estimated by the network miss all detail. Apparently the images trained upon are too similar. This results in the network finding a generic solution that works quite well for all images. It does not pay any attention any more to the details of for instance where a car exactly starts and ends. It is interesting to see if these abstract depth images are enough to initialize and improve the ORB-SLAM tracking accuracy.

Both the networks, fine-tuned on Kitti and on WEpod images, have been used in cooperation with ORB-SLAM on the WEpod sequences. The result is positive, ORB-SLAM manages to initialize almost immediately with the help of either network. The tracking accuracy of ORB-SLAM together with the CNN fine-tuned on Kitti images is better then of ORB-SLAM together with the CNN fine-tuned on the WEpod images. However, both versions lost tracking relatively quickly when compared to ORB-SLAM with the depth-estimating CNN on Kitti images. The problem appears to be that there still seems to be a scale drift over time. The tracking accuracy of ORB-SLAM with a depth-estimating CNN on WEpod images is quite worse then on Kitti images. There is still a gap between the tracking accuracy of ORB-SLAM on Kitti images and on WEpod images but it has been made much smaller.

The research done in this thesis has opened up a lot of other interesting research possibilities. The resulting combination of ORB-SLAM with a depth-estimating CNN still has a lot of room for improvements. First of all is it possible to improve the performance of the network fine-tuned on WEpod images. This can be done by using not only a larger set of images, but also a more diverse set of images. The network can be fine-tuned on more images taken from different scenarios in different environments. This will force the network to not learn this one basic abstract representation of the depth-maps but it will learn to pay more attention to the details.

Obtaining such a dataset of WEpod images together with ground-truth depth-maps is quite challenging. Another possibility of increasing the performance of the depth-estimating CNN on WEpod images is by looking further into the differences between WEpod and Kitti images. The performance of the Kitti fine-tuned CNN performing on WEpod images can be increased by making the WEpod images more similar to the Kitti images. This can only be done up to a certain extent as the Kitti images and WEpod images are from different countries, with different road layout, housing, nature etc.

Another possibility of increasing the performance of the depth-estimating CNN is by fine-tuning it on a mix of WEpod images and Kitti images. This way it can learn to create detailed depth images because of the Kitti images and learn the WEpod image type and scenarios because of the WEpod images. It would be interesting to investigate if this would result in better prediction of the WEpod images and influences the prediction of the Kitti images.

The setup of ORB-SLAM with the CNN could be extended to include other algorithms that can provide more information to ORB-SLAM to improve the tracking accuracy. Especially when the addition would be a neural network, it would give more possibilities. The two CNN's can be fused together into a smaller version. Such a cooperation between the CNN's could be one where they share the image encoding part and each have their own decoder part. They could also share information throughout each couple of layers and provide each other with useful information. This could improve the outcome of both networks.

An example of a CNN that could be added to ORB-SLAM and the depth-estimating CNN is a pose-estimating CNN. This pose-estimating CNN could use the images together with depth information from the depth-estimating CNN to estimate the pose between two images. This pose could then be used as a starting value for the optimization algorithms of ORB-SLAM. Such a pose estimating CNN already exists, such as in [57].

Other localizing algorithms based on GPS and lidar data on the WEpod could also be incorporated to work together with ORB-SLAM. The estimated pose from these algorithms can be used to set the starting value for optimization's within ORB-SLAM and constraints for the outcome of these optimization's. These constraints could be set based on the total scale of the scene, forcing ORB-SLAM to keep the same scale and eliminating the scale-drift problem that is still present in the depth-estimating CNN and ORB-SLAM combination on WEpod images.

The goal of improving the tracking accuracy of ORB-SLAM together with the depth-estimating CNN on WEpod images has been achieved. It has been proven that adding the depth-estimating CNN to monocular ORB-SLAM improves its tracking accuracy and that it solves the issue of monocular ORB-SLAM not initializing on WEpod images. This setup can be used together with the existing localization methods on the WEpod to improve the localization. This setup of ORB-SLAM with the network can not be used as the only localization method for the WEpod as it does not yet perform good enough. Multiple possibilities of improving this setup have been presented. This setup sets the next step for the WEpod being able to provide mobility to those who cannot drive and preventing human error caused traffic accidents.

5.2. Conclusion

This master thesis research has been done in cooperation with Robot Engineering Systems, a company that develops the WEpod. The WEpod is a self-driving vehicle capable of driving in mixed traffic. The WEpod is developed with the goal to improve the lives of many people. It's purpose is to bring the mobility benefit of a vehicle to people that are incapable of driving. Self-driving vehicles such as the WEpod can bring down the amount of road accidents that occur due to human errors.

The WEpod currently localizes itself on a pre-built map using GPS signals and lidar points. Unfortunately is the performance of these two not good enough. The WEpod is unable to safely navigate in between traffic or to plan for instance an overtake. The GPS signals can get interrupted or reflect off big objects. The 4-plane lidar localization algorithm is not robust enough due to the sparse and abstract representation of the surrounding it senses. The pre-built map is also created by a third-party and quite expensive. Therefore research is done into other methods of localization and mapping.

One of the solutions investigated for this localization problem is SLAM methods, especially monocular SLAM methods. There is a vision-based requirement because the costs of a camera is much smaller then the cost of for instance a 64-plane lidar. Unfortunately do state of the art monocular SLAM algorithms not perform adequate enough on images from a WEpod camera. The algorithms fail to initialize the surrounding scene correctly and thus result in a non existing tracking accuracy. This is where this thesis research fits in. The goal is to improve the tracking accuracy of the state of the art monocular SLAM algorithm ORB-SLAM, while keeping the whole setup monocular.

The initialization problem of monocular SLAM algorithms is a recognized problem. People have solved it by adding depth information to the inputs of the SLAM algorithm. The WEpod does not have any sensors equipped that can estimate the depth in its images. Therefore, the addition of ground-truth depth-maps to the inputs of ORB-SLAM has been investigated on the Kitti dataset. It can be concluded that the addition of depth-maps to ORB-SLAM does improve the tracking accuracy of ORB-SLAM. The WEpod sensors cannot create the depth-maps necessary, so a different solution needs to be found to obtain the depth-maps.

The possible solution investigated in this research is the use of a depth estimating Convolutional Neural Network (CNN). This CNN estimates the depth in monocular images and creates corresponding depth-maps. These depth-maps combined with the original image can be fed into ORB-SLAM to performs SLAM on. The network was originally trained on the NYU dataset which consists of indoor images and needs to be fine-tuned on images of road scenarios.

Fine-tuning has been done on data from the Kitti dataset, which consists of road scenarios. This CNN has been combined with ORB-SLAM. The tracking accuracy of this setup is better then monocular ORB-SLAM and worse then ORB-SLAM with ground-truth depth-maps. This shows that the addition of a depth estimating CNN to monocular ORB-SLAM does indeed improve the tracking accuracy. This whole setup also is still monocular as it only uses 1 camera image stream.

The CNN needs to be fine-tuned on a diverse set of images and depth-maps. Obtaining enough diverse depth-maps for the WEpod during this research was not possible. It was possible to obtain a relative small set of images with depth information and the CNN has been fine-tuned on this set. The network performed poorly as the set was not diverse enough. The CNN created basic abstract depth-maps without detail that did result in relatively good metrics. The CNN fine-tuned on Kitti images performed better on WEpod images than the CNN fine-tuned on WEpod images.

Both CNN's, fine-tuned on Kitti data and on WEpod data, have been combined with ORB-SLAM on WEpod images. Adding the CNN fine-tuned on Kitti data results in a better tracking accuracy then adding the CNN fine-tuned on WEpod data. Both do initialize correctly. This shows that adding a depth-estimating CNN solves the initialization problem and improves the tracking accuracy. It can also be concluded that the CNN does not need to be fine-tuned on the same dataset it performs on. Adding a CNN fine-tuned on a different dataset then predicting on can still improve the tracking accuracy of monocular ORB-SLAM.

This setup of a monocular SLAM method together with a depth-estimating CNN can be combined with the current localization methods of the WEpod. This could improve the robustness of the localization. The WEpod could navigate through the traffic more safely with this improved localization. This opens up a wide aspect of research possibilities by making it possible to for instance overtake objects. The improved localization adds to the development of the WEpod into the mobility supplying and safer self-driving vehicle it is destined to be.

The performance of monocular SLAM with the CNN is not robust and accurate enough to be the only localization method of the WEpod. It could be improved in multiple ways which could make this possible. One way of improving the setup is improving the result of the CNN. By fine-tuning the CNN on a larger and more diverse WEpod dataset, the quality of the estimated depth-maps can increase. The tracking accuracy of ORB-SLAM together with the CNN can also be improved by adding pose information. This pose information can be used as the starting values for the optimization algorithms in ORB-SLAM. The pose information can come from GPS data or a pose estimating CNN. There exist CNN's that estimate the pose difference between two images by using the images and the depth-maps. These depth-maps are already available through the depth-estimating CNN.

The use of the depth estimating CNN within the WEpod could also have many other benefits other then localization. A few examples of tasks done by the WEpod that could benefit from depth information are object-detection, scene segmentation and lane detection. The depth-estimating CNN could cooperate easily with the other networks by sharing information, resulting in improved performances of both the networks.

The goal of this thesis was to improve the tracking accuracy of monocular ORB-SLAM on WEpod images by adding a depth-estimating CNN. Monocular ORB-SLAM fails to initialize on only WEpod images. It has been proven that the addition of a depth-estimating CNN does indeed improve the tracking accuracy of monocular ORB-SLAM. The initialization problem on WEpod images is solved. This setup of ORB-SLAM with the CNN can be used to improve the robustness and accuracy of the localization of the WEpod. With this added accuracy and robustness in localization, the WEpod can navigate more safely in between traffic. It sets the next step in developing self-driving vehicles that not only improve the mobility of those that cannot drive, but also prevent traffic accidents caused by human error.

Bibliography

[1] Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Yangqing Jia, Rafal Jozefowicz, Lukasz Kaiser, Manjunath Kudlur, Josh Levenberg, Dandelion Mané, Rajat Monga, Sherry Moore, Derek Murray, Chris Olah, Mike Schuster, Jonathon Shlens, Benoit Steiner, Ilya Sutskever, Kunal Talwar, Paul Tucker, Vincent Vanhoucke, Vijay Vasudevan, Fernanda Viégas, Oriol Vinyals, Pete Warden, Martin Wattenberg, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. TensorFlow: Large-scale machine learning on heterogeneous systems, 2015. URL `https://www.tensorflow.org/`. Software available from tensorflow.org.

[2] Mariusz Bojarski, Davide Del Testa, Daniel Dworakowski, Bernhard Firner, Beat Flepp, Prasoon Goyal, Lawrence D. Jackel, Mathew Monfort, Urs Muller, Jiakai Zhang, Xin Zhang, Jake Zhao, and Karol Zieba. End to end learning for self-driving cars. *CoRR*, abs/1604.07316, 2016. URL `http://arxiv.org/abs/1604.07316`.

[3] Giorgio Grisetti Cyrill Stachniss, Udo Frese. Openslam. URL `http://openslam.org/`.

[4] Dalip and Vijay Kumar. Effect of environmental parameters on gsm and gps. *Indian Journal of Science and Technology*, 7(8), 2014. ISSN 0974 -5645.

[5] David Eigen and Rob Fergus. Predicting depth, surface normals and semantic labels with a common multi-scale convolutional architecture. *CoRR*, abs/1411.4734, 2014. URL `http://arxiv.org/abs/1411.4734`.

[6] David Eigen, Christian Puhrsch, and Rob Fergus. Depth map prediction from a single image using a multi-scale deep network. In Z. Ghahramani, M. Welling, C. Cortes, N. D. Lawrence, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems 27*, pages 2366–2374. Curran Associates, Inc., 2014.

[7] J. Engel, J. Sturm, and D. Cremers. Semi-dense visual odometry for a monocular camera. In *2013 IEEE International Conference on Computer Vision*, pages 1449–1456, Dec 2013. doi: 10.1109/ICCV.2013.183.

[8] J. Engel, J. Stückler, and D. Cremers. Large-scale direct slam with stereo cameras. In *2015 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 1935–1942, Sept 2015. doi: 10.1109/IROS.2015.7353631.

[9] Jakob Engel, Thomas Schöps, and Daniel Cremers. *LSD-SLAM: Large-Scale Direct Monocular SLAM*, pages 834–849. Springer International Publishing, Cham, 2014. doi: 10.1007/978-3-319-10605-2_54.

[10] Jakob Engel, Vladlen Koltun, and Daniel Cremers. Direct sparse odometry. *CoRR*, abs/1607.02565, 2016.

[11] Jakob Engel, Vladyslav C. Usenko, and Daniel Cremers. A photometrically calibrated benchmark for monocular visual odometry. *CoRR*, abs/1607.02555, 2016.

[12] Chris Engels, Henrik Stewénius, and David Nistér. Bundle adjustment rules. *Photogrammetric computer vision*, 2:32, 2006.

[13] Nolang Fanani, Alina Stürck, Matthias Ochs, Henry Bradler, and Rudolf Mester. Predictive monocular odometry (pmo): What is possible without ransac and multiframe bundle adjustment? *Image and Vision Computing*, 68(Supplement C):3 – 13, 2017. ISSN 0262-8856. doi: https://doi.org/10.1016/j.imavis.2017.08.002. Automotive Vision: Challenges, Trends, Technologies and Systems for Vision-Based Intelligent Vehicles.

[14] Olivier Faugeras and F Lustman. Motion and structure from motion in a piecewise planar environment. 02, 09 1988.

[15] Serena Yeung Fei-Fei Li, Justin Johnson. Course notes of cs231n: Convolutional neural networks for visual recognition, spring 2017.

[16] D. Filliat. A visual bag of words method for interactive qualitative localization and mapping. In *Proceedings 2007 IEEE International Conference on Robotics and Automation*, pages 3921–3926, April 2007. doi: 10.1109/ROBOT.2007.364080.

[17] C. Forster, Z. Zhang, M. Gassner, M. Werlberger, and D. Scaramuzza. Svo: Semidirect visual odometry for monocular and multicamera systems. *IEEE Transactions on Robotics*, 33(2):249–265, April 2017. ISSN 1552-3098. doi: 10.1109/TRO.2016.2623335.

[18] Ravi Garg, Vijay Kumar B. G, and Ian D. Reid. Unsupervised CNN for single view depth estimation: Geometry to the rescue. *CoRR*, abs/1603.04992, 2016.

[19] Andreas Geiger. The kitti vision benchmark suite. URL `http://www.cvlibs.net/datasets/kitti/`.

[20] Andreas Geiger, Philip Lenz, and Raquel Urtasun. Are we ready for autonomous driving? the kitti vision benchmark suite. In *Conference on Computer Vision and Pattern Recognition (CVPR)*, 2012.

[21] Ross B. Girshick, Jeff Donahue, Trevor Darrell, and Jitendra Malik. Rich feature hierarchies for accurate object detection and semantic segmentation. *CoRR*, abs/1311.2524, 2013.

[22] Ian J Goodfellow, Jonathon Shlens, and Christian Szegedy. Explaining and harnessing adversarial examples. *arXiv preprint arXiv:1412.6572*, 2014.

[23] Richard Hartley and Andrew Zisserman. *Multiple view geometry in computer vision*. Cambridge university press, 2003.

[24] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. *CoRR*, abs/1512.03385, 2015.

[25] D. Hoiem, A. A. Efros, and M. Hebert. Geometric context from a single image. In *Tenth IEEE International Conference on Computer Vision (ICCV'05) Volume 1*, volume 1, pages 654–661 Vol. 1, Oct 2005. doi: 10.1109/ICCV.2005.107.

[26] SAE International. Taxonomy and definitions for terms related to driving automation systems for on-road motor vehicles, 2016.

[27] A. Karpathy. What i learned from competing against a convnet on imagenet. `http://karpathy.github.io/2014/09/02/what-i-learned-from-competing-against-a-convnet-on-imagenet/`, 2015.

[28] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *CoRR*, abs/1412.6980, 2014. URL `http://arxiv.org/abs/1412.6980`.

[29] R. Kuemmerle, G. Grisetti, H. Strasdat, K. Konolige, and W. Burgard. g2o: A general framework for graph optimization. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, pages 3607–3613, Shanghai, China, May 2011. doi: 10.1109/ICRA.2011.5979949.

[30] Lubor Ladicky, Jianbo Shi, and Marc Pollefeys. Pulling things out of perspective. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 89–96, 2014.

[31] Iro Laina, Christian Rupprecht, Vasileios Belagiannis, Federico Tombari, and Nassir Navab. Deeper depth prediction with fully convolutional residual networks. *CoRR*, abs/1606.00373, 2016.

[32] Bo Li, Chunhua Shen, Yuchao Dai, Anton van den Hengel, and Mingyi He. Depth and surface normal estimation from monocular images using regression on deep features and hierarchical crfs. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 1119–1127, 2015.

[33] B. Liu, S. Gould, and D. Koller. Single image depth estimation from predicted semantic labels. In *2010 IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, pages 1253–1260, June 2010. doi: 10.1109/CVPR.2010.5539823.

[34] Fayao Liu, Chunhua Shen, and Guosheng Lin. Deep convolutional neural fields for depth estimation from a single image. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2015.

[35] Jonathan L Long, Ning Zhang, and Trevor Darrell. Do convnets learn correspondence? In *Advances in Neural Information Processing Systems*, pages 1601–1609, 2014.

[36] et. al. Lozano. Global and regional mortality from 235 causes of death for 20 age groups in 1990 and 2010: a systematic analysis for the Global Burden of Disease Study 2010. *The Lancet*, 380(9859):2095–2128, 2012. ISSN 0140-6736. doi: https://doi.org/10.1016/S0140-6736(12)61728-0.

[37] Aravindh Mahendran and Andrea Vedaldi. Understanding deep image representations by inverting them. *CoRR*, abs/1412.0035, 2014.

[38] M. Hossein Mirabdollah and Bärbel Mertsching. *Fast Techniques for Monocular Visual Odometry*, pages 297–307. Springer International Publishing, Cham, 2015. ISBN 978-3-319-24947-6. doi: 10.1007/978-3-319-24947-6_24.

[39] R. Mur-Artal, J. M. M. Montiel, and J. D. Tardós. Orb-slam: A versatile and accurate monocular slam system. *IEEE Transactions on Robotics*, 31(5):1147–1163, Oct 2015. ISSN 1552-3098. doi: 10.1109/TRO.2015.2463671.

[40] Raul Mur-Artal and Juan D. Tardós. ORB-SLAM2: an open-source SLAM system for monocular, stereo and RGB-D cameras. *CoRR*, abs/1610.06475, 2016.

[41] G Pasaoglu, D Fiorello, A Martino, G Scarcella, A Alemanno, A Zubaryeva, and C Thiel. Driving and parking patterns of european car drivers-a mobility survey. 2012.

[42] F. I. Pereira, G. Ilha, J. Luft, M. Negreiros, and A. Susin. Monocular visual odometry with cyclic estimation. In *2017 30th SIBGRAPI Conference on Graphics, Patterns and Images (SIBGRAPI)*, pages 1–6, Oct 2017. doi: 10.1109/SIBGRAPI.2017.7.

[43] E. Rublee, V. Rabaud, K. Konolige, and G. Bradski. Orb: An efficient alternative to sift or surf. In *2011 International Conference on Computer Vision*, pages 2564–2571, Nov 2011. doi: 10.1109/ICCV.2011.6126544.

[44] F. Schuster, C. G. Keller, M. Rapp, M. Haueis, and C. Curio. Landmark based radar slam using graph optimization. In *2016 IEEE 19th International Conference on Intelligent Transportation Systems (ITSC)*, pages 2559–2564, Nov 2016. doi: 10.1109/ITSC.2016.7795967.

[45] Ali Sharif Razavian, Hossein Azizpour, Josephine Sullivan, and Stefan Carlsson. Cnn features off-the-shelf: An astounding baseline for recognition. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR) Workshops*, June 2014.

[46] Nathan Silberman, Derek Hoiem, Pushmeet Kohli, and Rob Fergus. Indoor segmentation and support inference from rgbd images. *Computer Vision–ECCV 2012*, pages 746–760, 2012.

[47] Karen Simonyan, Andrea Vedaldi, and Andrew Zisserman. Deep inside convolutional networks: Visualising image classification models and saliency maps. *CoRR*, abs/1312.6034, 2013.

[48] Jost Tobias Springenberg, Alexey Dosovitskiy, Thomas Brox, and Martin A. Riedmiller. Striving for simplicity: The all convolutional net. *CoRR*, abs/1412.6806, 2014.

[49] Neville A. Stanton and Paul M. Salmon. Human error taxonomies applied to driving: A generic driver error taxonomy and its implications for intelligent transport systems. *Safety Science*, 47(2):227 – 237, 2009. ISSN 0925-7535. doi: https://doi.org/10.1016/j.ssci.2008.03.006. URL `http://www.sciencedirect.com/science/article/pii/S0925753508000441`.

[50] Christian Szegedy, Wojciech Zaremba, Ilya Sutskever, Joan Bruna, Dumitru Erhan, Ian J. Goodfellow, and Rob Fergus. Intriguing properties of neural networks. *CoRR*, abs/1312.6199, 2013.

[51] Keisuke Tateno, Federico Tombari, Iro Laina, and Nassir Navab. CNN-SLAM: real-time dense monocular SLAM with learned depth prediction. *CoRR*, abs/1704.03489, 2017.

[52] Brian Tefft. American driving survey. *AAA Foundation for Traffic Safety*, 2015-2016.

[53] John R Treat, NS Tumbas, ST McDonald, D Shinar, Rex D Hume, RE Mayer, RL Stansifer, and NJ Castellan. Tri-level study of the causes of traffic accidents: final report. executive summary.

[54] Shimon Ullman. *The interpretation of visual motion.* Massachusetts Inst of Technology Pr, 1979.

[55] Rui Wang, Martin Schwörer, and Daniel Cremers. Stereo DSO: large-scale direct sparse visual odometry with stereo cameras. *CoRR*, abs/1708.07878, 2017.

[56] Matthew D. Zeiler and Rob Fergus. Visualizing and understanding convolutional networks. *CoRR*, abs/1311.2901, 2013.

[57] Tinghui Zhou, Matthew Brown, Noah Snavely, and David G. Lowe. Unsupervised learning of depth and ego-motion from video. *CoRR*, abs/1704.07813, 2017.

[58] J. Zijlmans. Vision based slam on wepod data. Technical report, TUDelft, RES, 2017.

[59] J. Zijlmans. Vision-based slam with cnn estimated depth information. Technical report, TUDelft, RES, 2018.