Single view depth estimation from train images

Mémoire

Tesnim Hadhri

Maîtrise en génie électrique - avec mémoire
Maître ès sciences (M. Sc.)

Québec, Canada

Single view depth estimation from train images

Mémoire

Tesnim Hadhri

Sous la direction de:

Jean-François Lalonde, directeur de recherche

Résumé

L'estimation de la profondeur consiste à calculer la distance entre différents points de la scène et la caméra. Savoir à quelle distance un objet donné est de la caméra permettrait de comprendre sa représentation spatiale. Les anciennes méthodes ont utilisé des paires d'images stéréo pour extraire la profondeur. Pour avoir une paire d'images stéréo, nous avons besoin d'une paire de caméras calibrées. Cependant, il est plus simple d'avoir une seule image étant donnée qu'aucun calibrage de caméra n'est alors nécessaire. C'est pour cette raison que les méthodes basées sur l'apprentissage sont apparues. Ils estiment la profondeur à partir d'une seule image. Les premières solutions des méthodes basées sur l'apprentissage ont utilisé la vérité terrain de la profondeur durant l'apprentissage. Cette vérité terrain est généralement acquise à partir de capteurs tels que Kinect ou Lidar. L'acquisition de profondeur est coûteuse et difficile, c'est pourquoi des méthodes auto-supervisées se sont apparues naturellement comme une solution. Ces méthodes ont montré de bons résultats pour l'estimation de la profondeur d'une seule image. Dans ce travail, nous proposons d'estimer des cartes de profondeur d'images prises du point de vue des conducteurs de train. Pour ce faire, nous avons proposé d'utiliser les contraintes géométriques et les paramètres standards des rails pour extraire la carte de profondeur à entre les rails, afin de la fournir comme signal de supervision au réseau. Il a été démontré que la carte de profondeur fournie au réseau résout le problème de la profondeur des voies ferrées qui apparaissent généralement comme des objets verticaux devant la caméra. Cela a également amélioré les résultats de l'estimation de la profondeur des séquences des trains. Au cours de ce projet, nous avons d'abord choisi certaines séquences de trains et déterminé leurs distances focales pour calculer la carte de profondeur de la voie ferrée. Nous avons utilisé ce jeu de données et les distances focales calculées pour affiner un modèle existant « Monodepth2 » pré-entrainé précédemment sur le jeu de données Kitti.

Abstract

Depth prediction is the task of computing the distance of different points in the scene from the camera. Knowing how far away a given object is from the camera would make it possible to understand its spatial representation. Early methods have used stereo pairs of images to extract depth. To have a stereo pair of images, we need a calibrated pair of cameras. However, it is simpler to have a single image as no calibration or synchronization is needed. For this reason, learning-based methods, which estimate depth from monocular images, have been introduced. Early solutions of learning-based problems have used ground truth depth for training, usually acquired from sensors such as Kinect or Lidar. Acquiring depth ground truth is expensive and difficult which is why self-supervised methods, which do not acquire such ground truth for fine-tuning, has appeared and have shown promising results for single image depth estimation. In this work, we propose to estimate depth maps for images taken from the train driver viewpoint. To do so, we propose to use geometry constraints and rails standard parameters to extract the depth map inside the rails, to provide it as a supervisory signal to the network. To this end, we first gathered a train sequences dataset and determined their focal lengths to compute the depth map inside the rails. Then we used this dataset and the computed focal lengths to finetune an existing model "Monodepth2" trained previously on the Kitti dataset. We show that the ground truth depth map provided to the network solves the problem of depth of the rail tracks which otherwise appear as standing objects in front of the camera. It also improves the results of depth estimation of train sequences.

Table des matières

Liste des tableaux

Liste des figures

***To my mother and father, Salwa and Lotfi**, who have always loved me unconditionally and taught me to work hard for the things that I aspire to achieve, for their endless support, for their kindness, for their encouragement during difficult moments.*

***To my lovely sisters Sirin and Elae**, who have never left my side, I value all of your inputs.*

***To my aunt, Samira**, who, although is no longer with us, always has been an inspiration by her unlimited support, help, and encouragement. You have been a mother to me.*

***To my dear husband, Wajih**, for your patience, love, friendship and humour. You have always been a source of support and encouragement during different challenges of both school and life. I am truly thankful for having you in my life.*

Acknowledgments

I would like to express my special gratitude to my supervisor Jean-François Lalonde whom support and unlimited patience and enthusiasm helped fuel me during all the challenges I encountered.
My sincerest thanks to my supervisor Benoit Debaque from the Thales group whose stimulating suggestions helped me to coordinate my project.
Above all, I thank ALLAH, for giving me the strength to face all challenges.

Introduction

Depth prediction is the task of computing the distance of different points in the scene from the camera. Knowing how far away a given object is from the camera would make it possible to understand its spatial representation. Depth is a crucial component in different applications. Tasks such as background removal, removing fog from scenes, portrait mode in cameras, are best when processed as a function of depth as shown in fig. 0.1 and fig. 0.2.

Portrait Mode takes images with a shallow depth of field, focusing on the primary subject and blurring out the background for a professional look. It does this by using machine learning to estimate how far away objects are from the camera, so the primary subject can be kept sharp and everything else can be blurred.

FIGURE 0.1 – Portrait mode examples using google pixel camera. It is possible to tell the background from the foreground by estimating how far objects are from the camera. Having the depth estimated makes it possible to keep the object sharp and neat and blur the background. Image taken from [2].

In robotics, localization and navigation are very important for the robot to determine its own spatial position, and have the ability to determine how far an object is away from it and then decide its navigation trajectory, which is why it usually requires a depth map of its surroundings as shown in fig. 0.3. Self-driving vehicles also need a depth map of the surroundings for the car to be able to localize itself, as well as be aware of objects and people in the scene.

(a) (b)

FIGURE 0.2 – Example of removing fog from images [31]. The thickness of the fog present in the picture is non-uniform and usually depends on depth. For instance, the fog surrounding objects far from the camera will appear denser in images than fog surrounding objects close to the camera, which is why depth is incorporated in haze removal techniques. Input image in left row (a), output image in right row (b). Images taken from [31].

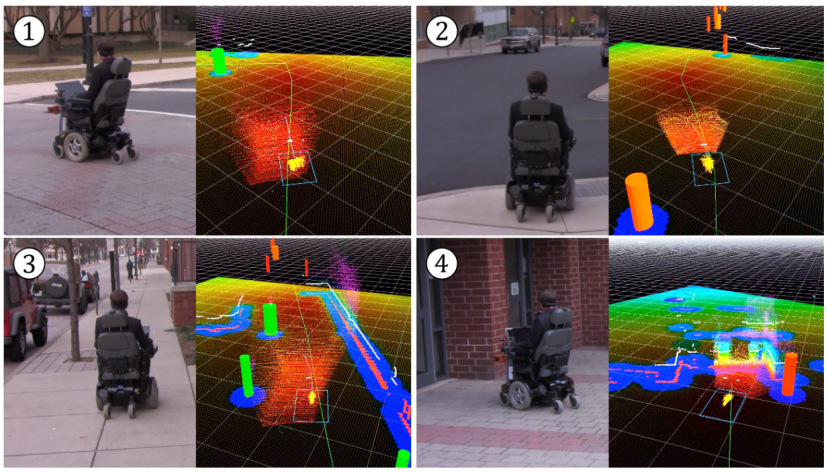


FIGURE 0.3 – Autonomous navigation of a wheelchair [25]. To navigate, the chair needs to determine its spatial position by mapping the depth of the scene. Video frames are on the left, data visualisation including 3D particle clouds are present on the right. Image taken from [25].

Augmented reality (AR) is a Technology which enhances the interactions between the real world and the virtual world by including digitized objects in the real-world environment, usually used in gaming, medicine -notably medical training and surgeries-, learning, tourism and different other domains. Depth maps in these applications are needed in order to have a more natural look of the inserted objects, avoid visual artifacts and offer a visual satisfaction to the user. Fig. 0.4 shows an example of of AR application developed using depth estimation.

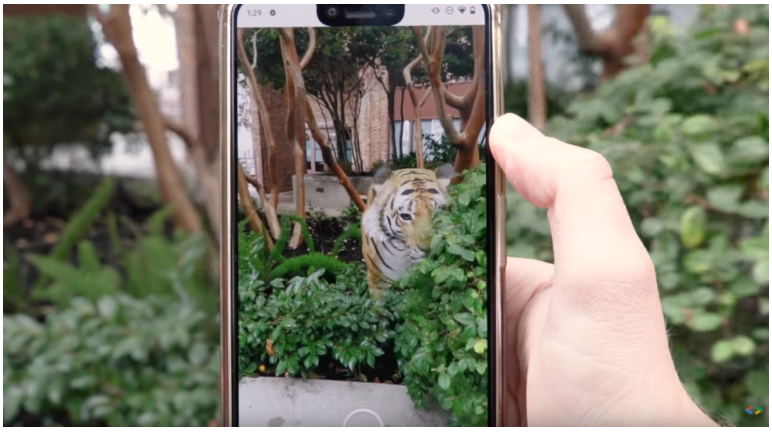
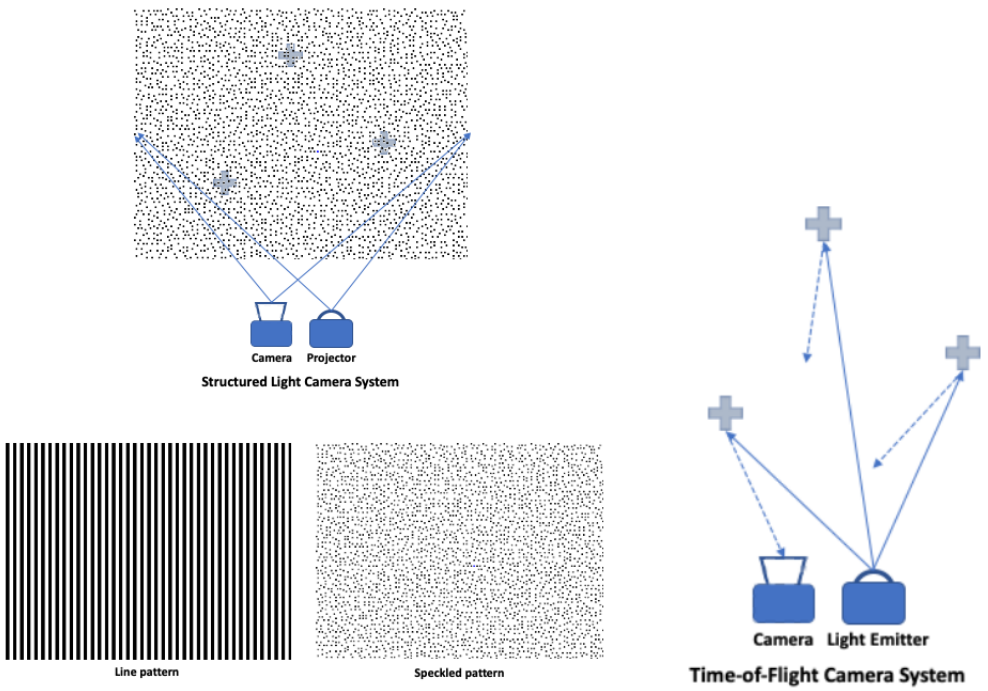


FIGURE 0.4 – Example of AR application using ARcore Depth API from google, an algorithm for depth estimation from a single RGB image using depth-from-motion [8], it helps as shown in the picture, to occlude an object, a lion in this case, in a very realistic way, as is capable to add some occlusion (lion hidden behind a tree in this picture). Image taken from this video.

To retrieve a depth map, we could use active remote sensing such as Kinect, or Lidar. Microsoft Kinect is an active sensor which maps depth by projecting light into the scene, depending on the Kinect version it either uses structured light or time of flight techniques as shown in fig. 0.5. Structured light technique sends a pattern of light in the scene, captures it using a camera and computes the depth map as function of the pattern warped on the object to provide a dense depth map of its surrounding environment. Kinect is not a good fit to map depth for outdoor environments, because of its limited light emission range and its sensitivity to the sun. Lidar is also an active sensor that uses the time of flight method to measure the depth. It sends beams of light and computes how long it took for the beam to hit an object and reflect back to a light detector. Lidar however outputs a sparse depth map. There are also other active sensors which, instead of sending a beam of light, rely on ultrasound to compute depth maps. The main limitation of active sensors is that they require a large amount of energy to operate. This limitation led the researchers to also focus on passive sensors, such as cameras. Different algorithms are proposed since the 90s to solve the depth estimation tasks, relying on stereo vision and stereo matching. [7] provides an overview of popular methods.

Extracting the depth map using stereo pairs is also difficult. To have a stereo pair of images, we need a calibrated pair of cameras. However it is simpler to have a single camera as no calibration or synchronization is needed.



(a) (b)

FIGURE 0.5 – Example of techniques used by active sensors to retrieve depth values. (a) represents the structured light technique which which sends light patterns onto the scene and captures it, (b) is the time-of-flight technique which sends a beam of light and computes the depth in function of how long it took the beam to touch an object and reflect back. Image taken from [20].

For this reason learning based methods have shown up, and are recently gaining a lot of attention in research since you can be independent to stereo images and estimate depth from monocular images. However it is an ill posed problem as only a single view image is provided. There are also other challenges, such as reflective surfaces, occlusion and dynamic objects, along with information that could be missing or hard to extract such as the focal length of the camera. Using deep learning, depth estimation could be classified in two main categories. First, supervised methods which rely on ground truth for training. These methods vary according to datasets : some use real world datasets, while others use virtual datasets. Second, self-supervised methods where infomtaions are usually extracted implicitly using the geometry of the scene or explicitly using only learning based techniques.
Different datasets are therefore present to support learning based techniques to estimate depth. These datasets usually come in different categories : we find datasets for indoor environments and datasets for outdoor environments. We also find synthetic datasets besides the real world datasets. There are a good number of datasets to support autonomous vehicle navigation. Unfortunately not all domains have gained the same attention as the self driving car. For example, until now there is only one dataset provided for trains, which itself does not

support all tasks since it was mainly developed for segmentation tasks. Due to this limitation, rails appear as standing objects in a lot of depth estimation methods.

In this work we are specifically interested in the estimation of depth maps for images taken from the train drivers point of view. To work on depth estimation specifically for trains, we started by developing a dataset for trains which includes different train sequences, their predicted focal lengths, and a part of the depth ground truth which could be used as a supervisory signal to help train the network. To achieve this goal we set the following steps. We first chose train sequences from a train enthusiasts website [28]. We then calibrate the camera of each sequence using the rails themselves. In order to validate the estimated focal length we will compare an estimated distance of a visible object in the sequence with the same object's distance in calibrated overhead imagery. Second, based on the predicted focal lengths of different sequences, we will compute the depth map inside the rails for the corresponding extracted frames. And finally, we will finetune monodepthV2's [16] network with our generated dataset. During the finetuning process we will conduct different experiments to see the influence of both ground truth depth maps and focal lengths during the training phase and the inference phase.

In chapter 1 we will present previous methods and different datasets used for the depth estimation task. In the second chapter, we will describe the process used for the dataset generation, this dataset will be generated specifically for depth estimation for train sequences. In chapter 3 we will follow with the network finetuning experiments using the generated dataset. We will explain different experiments and present their results. And finally a conclusion will sum up different steps held during this project, its limitations and possible future work.

Chapitre 1

Related work

1.1 Introduction

Perception of the objects' depth in a scene is a very important problem in computer vision as depth information is a key component in different domains such as robotics, AR, autonomous driving and several others. Depth can be obtained using dedicated sensors such as Kinect or Lidar which project light in the scene in order to obtain precise depth measurements. In this work we will be focusing on methods that estimate the depth from a single input image captured by a conventional camera. Early image-based methods used stereo or multiple cameras to compute depth maps for a corresponding image. However, relying only on epipolar geometry requires a lot of difficult-to-acquire informations to be provided such as the baseline measure and the focal length which sometimes could possibly be not available. For this reason, several others looked for alternative methods to retrieve depth maps for a single image. Those alternative methods can be either supervised methods where they rely on ground truth depth to train a deep neural network, self supervised methods which rely on stereo images or temporal frames to retrieve information. These methods can also follow a joint learning architecture where they train for both depth and pose or flow motion, as these information can be beneficial for depth retrieval. One among used methods is the depth map completion task which relies on partially known depth maps to generate a dense depth map for a given input. In Section 1.2 we will briefly cover some depth estimation methods, then we will present a set of relevant datasets.

1.2 Depth estimation

We will briefly introduce some depth estimation techniques. We begin with supervised methods, and follow with self-supervised methods.

1.2.1 Supervised methods

Several previous methods have solved the problem in a supervised learning setting, where both the color image and its corresponding ground truth depth map are used to train a neural network. Here the task of the learning algorithm is to discover a relationship between the input and its given depth values.

Usually, these methods differ either in the network architecture or their loss function. [21] uses an end-to-end convolutional neural network to solve for depth estimation using a single RGB input image, whereas [11] handles the problem of depth estimation as an ordinal regression problem by discretizing depth using a space increasing discretization strategy.

Another method focuses on solving the problem of spatial resolution of this task [19] since-previous methods usually suffer from hazy boundaries. It proposed an improvement of the network architecture as well as the loss function. They proposed an additional measurement specifically to evaluate depth edges and quantitative results were greater than other methods. Visually, edge boundaries of their depth maps also looked much better.

Yet, due to the lack of dense ground truth depth maps, some supervised methods solved the problem as a depth completion problem where they generate dense depth maps from given sparse depth maps [4] [23].

These learning based methods require the collection of ground truth depth maps which is a difficult and time consuming task. Therefore, some weakly supervised methods rely on synthetic data for training as an alternative. For example [40] uses synthetic data with its corresponding depth map for the depth task. During training, it uses a GAN to translate the synthetic image into the real image as a first stage followed with a depth estimator architecture. At inference time, only the real input data is used.

Some methods use synthetic data for training, notably [1] which uses style transfer and adversarial training to predict depth on real-world data.

1.2.2 Self-supervised methods

Ground truth depth maps are difficult to acquire for two main reasons. First, it is difficult to acquire a sufficient amount of ground truth data to train a network for each task, second capturing depth is an expensive process. One major problem is that depth sensors are limited. For instance, kinect has a limited depth range and it thus is difficult to have a depth map of the outside environment using this sensor. Lidar sensors provide only sparse depth maps due to their limited angular sampling rate. For this reason, researchers started using self supervised methods where ground truth depth maps are not needed for training.

Self supervised methods use a different way to train depth estimation networks which relies mainly on reconstruction. Network inputs are either stereo sets of images or monocular temporal sequences, or both. In this case, depth is reconstructed by minimising the reconstruction error of a stereo pair or a temporal pair of images.

Another method [13] relies on trying to warp the right image into the left image from the estimated depth. The little motion between the image pair makes it possible to the network to explicitly retrieve the relationship between them which actually is the inverse depth. A follow up on the same approach [15] has built on [13] adding a left-right consistency loss.

Some methods rely on joint learning to solve for depth estimation, notably [38] which uses a network architecture that solves for both odometry and depth estimation by using stereo sequence, which allows the network to learn information from temporal frames and the stereo pair.

Other methods such as [32] solves for both depth and structure from motion trains a convolutional network using uncalibrated image pair. The main goal of the network during training is to learn matching.

Monodepth2 [16] uses stereo sequences, with which it would be possible to have information from the stereo pair as well as the temporal frames. Aside from that, it improves the approach with a minimum reprojection loss to solve the problem of occlusions. Monodepth2 [16] introduces a technique called auto-masking, where pixels that appear to be moving with the same velocity as the camera are excluded from the full loss module. This technique also helps eliminate static scenes from the loss function. further description about this work will be explained in the next section.

1.2.3 Monodepth

In this work we used [16]'s model for finetuning what is actually a follow up on [15]'s work. This is why this detailed description is presented about both [15] and [16].

One of the most used self supervised methods is "Monodepth" [15] as it has shown a great improvement using stereo pairs. The model learns to figure out a single image depth estimation without ground truth provided to the network. "Monodepth" relies on explicitly extracting the disparity while trying to reconstruct the left image from the right image by warping. In epipolar geometry, disparity is inverse of depth.

The main contribution of [15] is in their loss function explained in fig. 1.1 which is a combination of three different modules. The first module is the appearance Matching Loss, which, as its name shows, compares the appearance of the source image and the target image, the appearance matching loss in [15] was inspired from [17]. The second module is the disparity

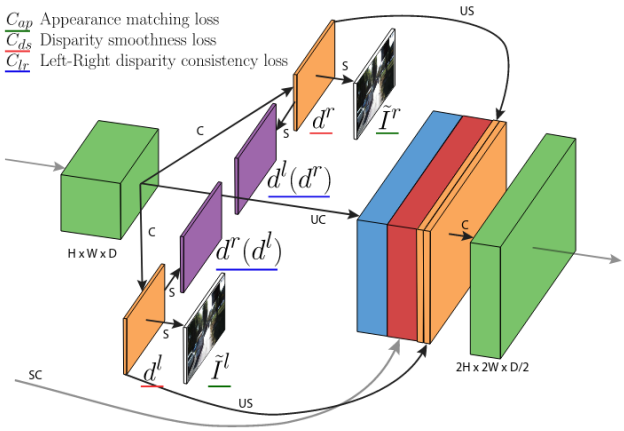


FIGURE 1.1 – Loss module from [15]. It includes both left and right disparity maps shown in orange as well as warped disparity maps in purple and the network's stereo input images. Appearance matching loss underlined in green, disparity smoothness loss in red and left-right disparity consistency loss in blue. The loss is applied every time at each stage, noted C, UC, S, US and SC respectively for Convolution, Up-Convolution, bilinear Sampling, Up-Sampling and Skip Connection. Image taken from [15].

smoothness loss, which forces the disparity map to be smooth and prevent discontinuities. The third module is the Left-Right Disparity Consistency Loss, for which the network is trained to predict both left and right disparity maps although the only source image is the left view, [15] takes the right disparity map, warp it into the left disparity map and computes the difference between them. This way both left and right disparity maps produced by the network will be coherent.

Total training loss

"Monodepth"'s total loss at each output scale is described as follows :

$$C_s = \alpha_{ap}(C^l_{ap} + C^r_{ap}) + \alpha_{ds}(C^l_{ds} + C^r_{ds}) + \alpha_{lr}(C^l_{lr} + C^r_{lr}), \tag{1.1}$$

where C_{ap} is the appearance matching loss which helps the constructed images to match the input image, C_{ds} is the smoothness loss, and C_{lr} is the left right consistency loss. Note that l and r in the loss label is to differentiate between left and right loss.

Appearance matching loss

Monodepth [15] computes the appearance matching loss using a combination of $L1$ and SSIM [33] inspired by [39].

$$C_{ap}^{l} = \frac{1}{N}\sum_{i,j} \alpha \frac{1 - SSIM(I_{ij}^{l}, \tilde{I}_{ij}^{l})}{2} + (1-\alpha) \parallel I_{ij}^{l} - \tilde{I}_{ij}^{l} \parallel . \tag{1.2}$$

where I_{ij}^{l} is the input image, $\tilde{I}_{ij}^{l}$ is the reconstructed image and N is the number of pixels. α is set to 0.85 according to the experiments presented in the paper.

Disparity smoothness loss

$$C_{lr}^{l} = \frac{1}{N}\sum_{i,j} \mid \partial_x d_{ij}^{l} \mid \exp^{-\|\partial_x I_{ij}^{l}\|} + \mid \partial_y d_{ij}^{l} \mid \exp^{-\|\partial_y I_{ij}^{l}\|}, \tag{1.3}$$

where N is the number of pixels, ∂d is the disparity gradients and I_{ij} is the input image.

Left-right disparity consistency loss

$$C_{lr}^{l} = \frac{1}{N}\sum_{i,j} \mid d_{ij}^{l} - d_{ij+d_{ij}^{l}}^{r} \mid, \tag{1.4}$$

Where d^{l} is the left disparity map and d^{r} is the right disparity map. This loss forces the left-view disparity map to be equal to the projected right-view disparity map as explained in the paper [15].

1.2.4 Monodepth2

"Monodepth2" [16] is a followup on "Monodepth". Its main goal is also to estimate depth using a single input image. Apart from training using a stereo pair, it brings the choice to also train with monocular sequences. Since monodepth also trains for camera pose estimation as an auxiliary task, it gives the user the flexibility to either train using stereo pairs, monocular sequences or both.

It can thus be trained either using both or using one of them. For this purpose, aside from training for depth prediction, they also train for pose estimation.

The model of [16] is composed of two main networks shown in fig. 1.2 : a depth estimation network and a pose estimation network. At training time, [16] minimises a photometric re-projection error using the target image $I(t)$, and the source image which is I in the stereo training, both $I(t-1)$ and $I(t+1)$ sequence frames in the monocular training, and the three inputs altogether in the monocular stereo training.

The photometric error is used between the warped target image and the source image. To warp the target image, [16] uses the depth map predicted by the depth network, camera intrinsics, and the pose which is predicted by the pose network. [16] uses edge aware smoothness loss on the predicted disparity map to smooth and prevent noise and discontinuities in depth maps edges. It also uses a full-res multi-scale loss presented in fig. 1.2 where they compute the loss at different scales to reduce artifacts.

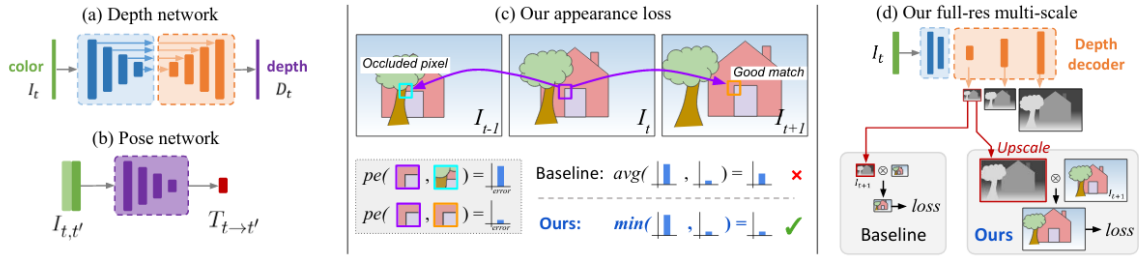


FIGURE 1.2 – Different monodepthv2 components, depth and pose networks' architectures in (a) and (b), minimum reprojection loss in (c) and a full-res multi-scale in (d). Image taken from [16].

Similar to [15], [16] uses a photometric reprojection loss as shown in 1.5 and 1.6,

$$L_p = \sum_{t'} pe(I_t, I_{t'\to t}), \tag{1.5}$$

$$I_{t'\to t} = I_{t'} \langle proj(D_t, T_{t'\to t}, K), \tag{1.6}$$

where I_t and $I_{t'}$ are respectively the source image and the target image, D_t is the predicted dense depth map and K is camera intrinsics matrix. $proj()$ are the 2D coordinates of the projected depth D_t. pe is the photometric reconstruction error function as shown in 1.7.

$$pe(I_a, I_b) = \alpha \frac{1 - SSIM(I_a, I_b)}{2} + (1 - \alpha) \parallel Ia - I_b \parallel, \tag{1.7}$$

where I_a and I_b are respectively the source image and the target image, α is set to 0.85 according to the paper.

Per-pixel minimum reprojection loss

The main contribution in the reprojection loss function is shown in eq. 1.8, where rather than average the loss over all source images, they simply take the minimum,

$$L_p = \min_{t'} pe(I_t, I_{t' \to t}) \tag{1.8}$$

Auto-Masking Stationary Pixels

Per pixel mask is applied to avoid stationary pixels contaminating the loss as shown in eq. 1.9

$$\mu \left[\min_{t'} pe(I_t, I_{t' \to t}) < \min_{t'} pe(I_t, I_{t'}) \right] \tag{1.9}$$

Final loss

We finally combine all losses to have the total loss shown in eq. 1.10

$$L = \mu L_p + \lambda L_s, \tag{1.10}$$

where λ is set to 0.001 according to the paper.

1.3 Data sets

Several datasets are provided for either depth completion or depth estimation tasks. There are datasets for the indoor environment and others for the outdoor environment. Some methods test their models on both categories. Although there exist several benchmarks for the depth estimation task, no dataset focuses on solving the depth estimation task for train navigation. We can divide datasets in two main categories : synthetic and real. There exists a variety of datasets for both indoor and outdoor environments for both the synthetic and real cases, however there are no suitable datasets in our scenario, which is estimating depth for train sequences from a single image. The following section presents a brief description for some available datasets for depth estimation tasks.

1.3.1 Real world datasets

NYU-Depth V2 [26] is a dataset of video sequences of different indoor scenes, it has both RGB images and its corresponding depth retrieved from Microsoft kinect.

ScanNet [6] is also an indoor dataset but aside from depth estimation it is also used for other tasks such as 3d object classification and semantic voxel labelling. It features over 1500 indoor scenes, its 3D camera pose annotations, surface reconstructions and instance-level semantic segmentation.

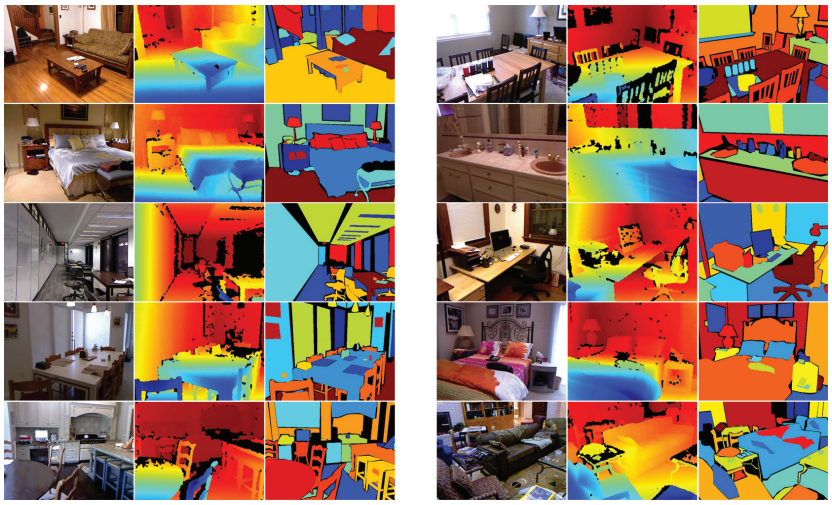


FIGURE 1.3 – Examples of NYU dataset, RGB image examples, raw depth image and the class labels from the dataset, Image taken from [29]

FIGURE 1.4 – Scannet dataset [6] examples. reconstructed 3D structures with its instance-level object annotations, each instance is presented with a different color. Image taken from [6].

Sun3D [35] SUN3D : provides RGB-D scenes with annotated objects and camera pose for large scale indoor environments. The dataset itself however, is reconstructed mainly using structure from motion technique (SFM).

(a) (b)

FIGURE 1.5 – Sun3d 2D view-based (a) and 3D place-centric (b) examples, images taken form [35].

Kitti [14] is a dataset of outdoor environments. It was developed as a benchmark for different computer vision tasks such as optical flow, visual odometry, single image depth estimation, depth completion, semantic segmentation, etc. Kitti sequences have been recorded with the AnnieWay mobile platform which includes two high-resolution color and grayscale video cameras, velodyne laser scanner for ground truth depth maps and a GPS/IMU localisation system. For the purpose of solving depth estimation and depth completion tasks, Kitti features 93 000 RGB images and their corresponding sparse Lidar depth maps. Kitti dataset image examples are shown in fig .1.6.

Cityscapes [5] is also a dataset for outdoor environments, it has stereo sequences of fifty different cities in different weather conditions ans corresponding GPS coordinates and ego-motion form vehicle odometry. Cityscapes focuses on different computer vision tasks such as semantic understanding for outdoor scenes, and object detection. Examples of Cityscapes dataset are shown in fig . 1.7.

1.3.2 Synthetic datasets

Interiornet [22] is a synthetic dataset for indoor environments. It features millions of objects and difference scenes with their corresponding ground truth depth. This dataset also comes

FIGURE 1.6 – Two frames example of Kitti dataset. Images taken from [27].

with an interactive simulator to help create monocular and stereo camera trajectories from a given layout database or different 3D scenes.

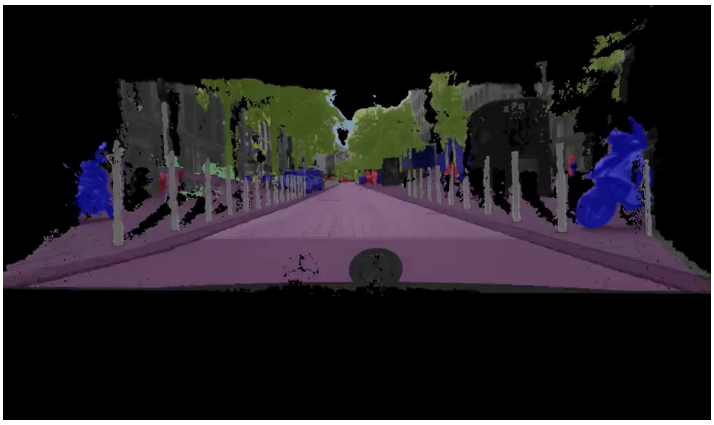
Vkitti [12] is the synthetic version of Kitti dataset. It is created with the Unity game engine and recreates the virtual version of Kitti sequences along with their ground truth depth. Vkitti recently published its latest version Vkitti2 [3]. Examples of Vkitti dataset are shown in fig .1.8.

Just like Vkitti, Synscapes [24] is the virtual version of cityscapes. It was generated using a photorealistic rendering technique. As it was created to be similar to cityscapes, it is used for the same tasks cityscapes was created for, such as for example, depth estimation and semantic segmentation.

4D light field [18] introduces 24 synthetic scenes with their corresponding depth ground truth. It provides 12 other scenes which are not part of the benchmark but are used for the purpose of evaluation. Example of 4D Light Field dataset is shown in fig . 1.9.

a

b



c

FIGURE 1.7 – Example of Cityscapes input image and GPS position visualisation in (a), finely annotated image in (b) and sparse depth map in (c). Images taken from [30].

FIGURE 1.8 – Examples of different kitti [14] and Vkitti [12] images, original kitti in top, the first version of virtual kitti in the second row and the second version of virtual kitti (Vkitti 2) in the last row. Images are taken from [10]

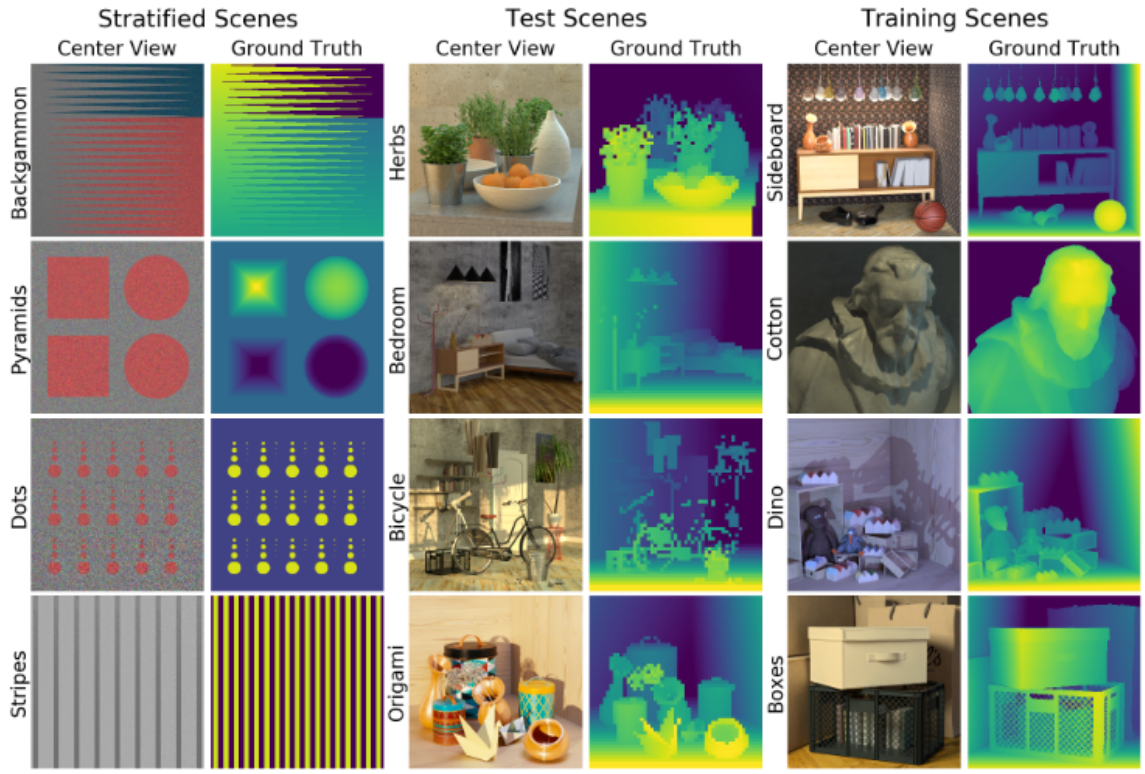


FIGURE 1.9 – 4D Light Feild [18] dataset example. Images are taken from [18].

1.4 Train datasets

WildDash [36] is a dataset for driving scenarios of different vehicles in different weather conditions and different environments and countries. WildDash's purpose is to provide a benchmark for different segmentations tasks. We are particularly interested in RailSem19 which is a sub-dataset of the Wilddash dataset.

RailSem19 [37] offers sequences from the driving perspective of trains and trams. It includes 8500 images taken from the rail vehicle. It provides semantic annotations for each frame. RailSem19 is the only dataset that is concerned about trains and trams. Unfortunately, it only provides annotations for segmentation tasks.

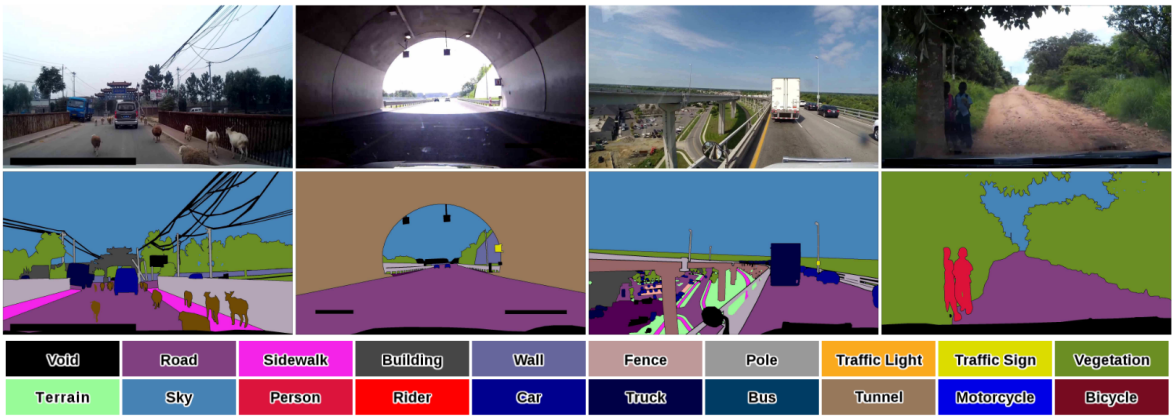


FIGURE 1.10 – Example of wildDash [36] dataset, input image, annotated dense ground truth and color legend. Images taken from [36].

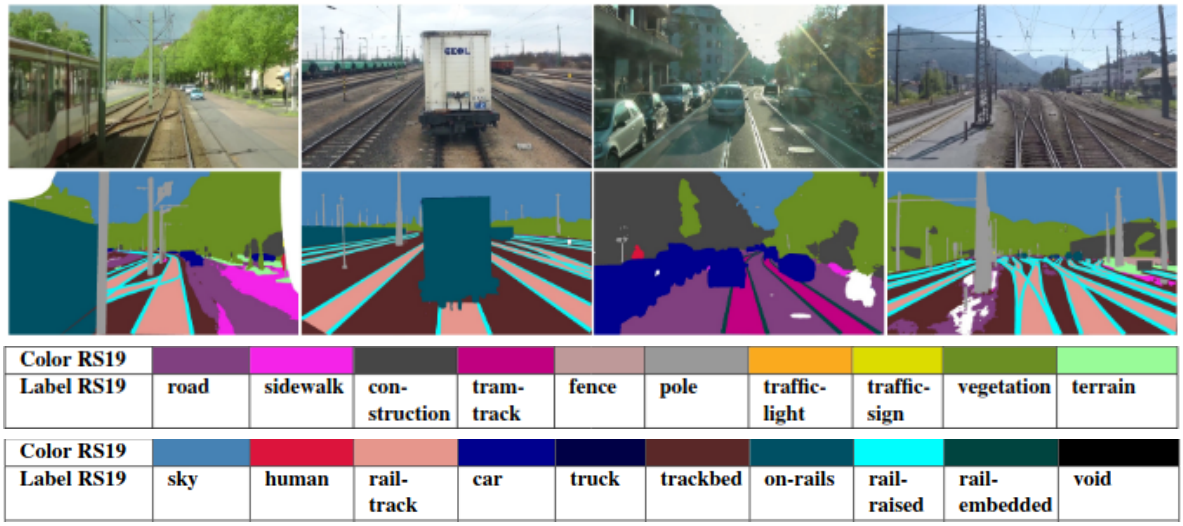


FIGURE 1.11 – RailSem19 [37] dataset example, input image, annotated dense ground truth and color legend. Images taken from [37].

Chapitre 2

Dataset generation

2.1 Introduction

As we have seen in the first chapter, data sets of outdoor environments with Lidar ground truth, stereo images and/or sparse depth maps retrieved from depth sensors are available, but the vast majority of them target autonomous car driving scenarios. However, it is quite difficult to find datasets for other domains such as train navigation. For this purpose, in this chapter we will explain the process used to generate navigating trains dataset that can be used to train depth estimation algorithms. Supervised methods have shown promising results in depth estimation from a single RGB image, yet as ground truth depth maps are difficult to acquire, we tried to look for an alternative way to provide a supervisory signal to the network.

One way to provide that is to rely on the geometric constraints of the scene which will allow us to retrieve part of the depth map and use it in the training phase. In addition we can also rely on the rails dimensions in the scene to retrieve part of the depth map of an RGB image. Depth information will be extracted mainly from geometric constraints of a moving camera in a mostly static scene and rails standard dimension and geometry. In this chapter, we will explain how we used the rails standard parameters to extract the focal length of the camera in a specific sequence and then use it to compute a depth map inside the rails region to use it as ground truth during the training process.

2.2 Data video sequences

The sequences we used for this data set were downloaded from the "Rail cabrides" website [28], which includes several navigating train sequences that have been recorded in different countries. Rail cabrides includes more than 8000 video sequences, mostly recorded from the driver's cabin. This collection includes different types of rail vehicles, including : metro, train,

tram, subways. Recordings are available in different weather conditions (rain, snow, fog, sunshine, cloudy), and also available underground. Sequences that were chosen for our dataset are metro sequences recorded either in town or country side of the Netherlands exept for the last sequence which is a train sequence. Sequences lengths are generally between 15 to 40 minutes as shown tab. 2.1. Note that we omitted the frames of the trains navigating inside tunnels.

Example image	Fps	Duration	Location	URL
	25	18 minutes 57 seconds	Haarlem - Amsterdam	Link1
	30	32 minutes 14 seconds	Haarlem - Alkmaar	Link2
	30	1 hour 18 minutes 20 seconds	Apeldoorn -Almelo - Enschede	Link3
	25	43 minutes 11 seconds	Amersfoort - Schothorst - Amsterdam	Link4
	30	27 minutes 26 seconds	Uitgeest - Beverwijk - Haarlem	Link5
	30	39 minutes 31 seconds	Rotterdam - Utrecht	Link6
	30	26 minutes 31 seconds	Zwolle - Nijverdal West	Link7

TABLE 2.1 – Informations about sequences present in the dataset. Example image, fps, duration, location and direct link to the sequence.

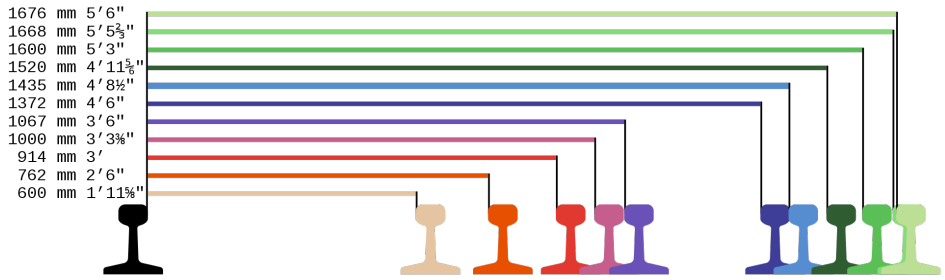
2.3 Estimating the camera focal length from rails

Standard parameter of rail tracks made it possible to extract the ground truth map, however it is only possible to extract the ground truth inside the railroad tracks.

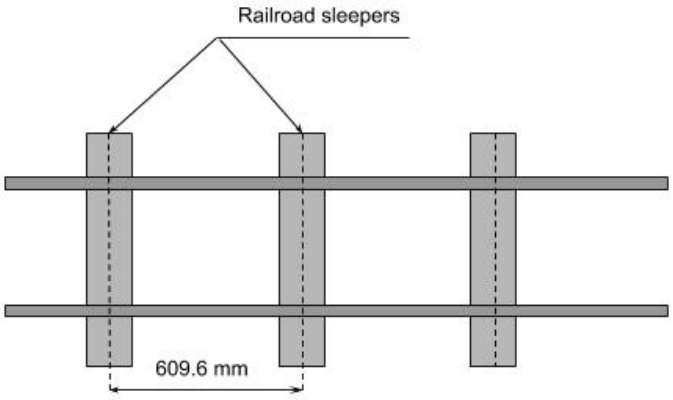
To extract the ground truth depth for video sequences we will first compute the camera focal length of each sequence in the dataset using standard gauge rail parameter and geometry constraints. Then having the camera focal length will allow us to compute the depth of each frame in the corresponding sequence. Finally we will validate each focal length using manually identified objects visible in both the video frame and calibrated overhead imagery.

2.3.1 Rails parameters

Two important parameters are used in the depth map computation. The first is rail gauge, also called track gauge, and is the measurement between the two inside faces of the rails (see fig. 2.1). The other parameter is the distance between two cross ties as shown in fig. 2.1, the cross tie or also called railroad tie (or rail road sleeper) is the rectangle connecting two railroad tracks and is usually made from either wood or concrete. Values in rail transport constructions differ according to the material of the railroad tie. Note that the values shown in both fig. 2.1 and fig. 2.1 are the values for concrete railroad ties.



(a)



(b)

FIGURE 2.1 – Railroad parameters. Different rail gauge values [34] with concrete railroad ties in (a). The most common rail gauge value is the standard value. Note that values change according to the material of the railroad tie which could be either wood or concrete. Standard concrete value between rail road sleepers in (b).

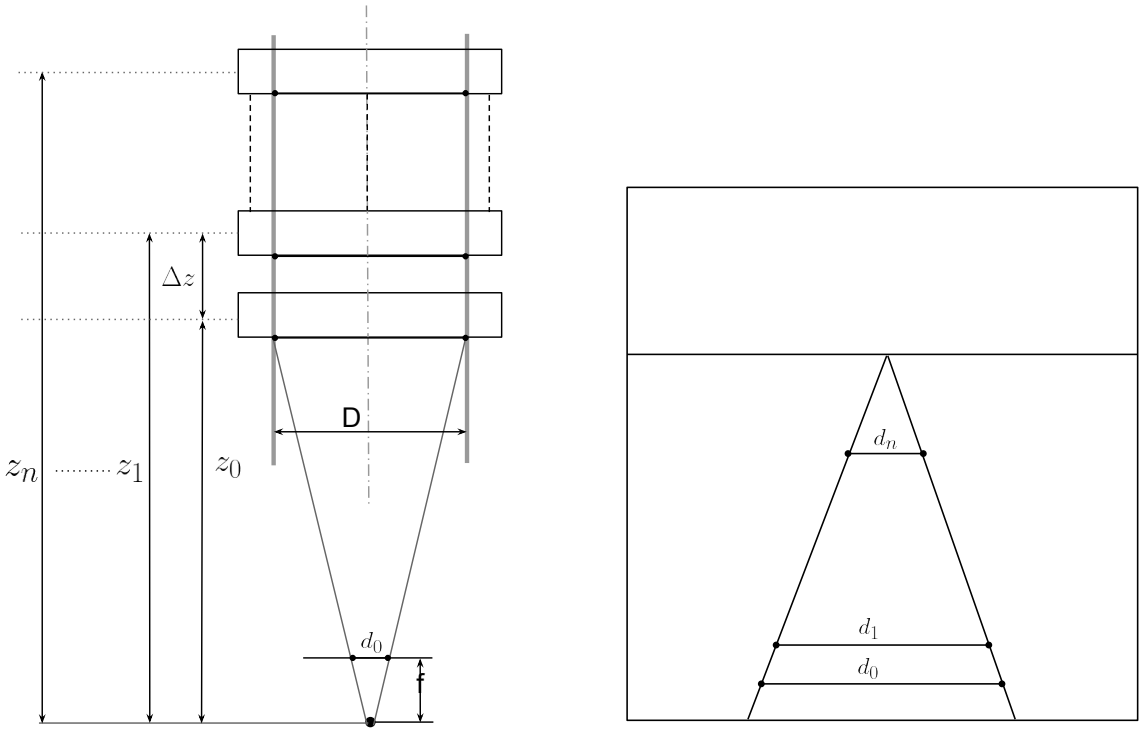
2.3.2 Computing the camera focal length

To generate the train depth map data set, we start by computing the camera focal length f for each one of the sequences. Note that we assume a pinhole model, thus there is no lens distortion effects.

The problem is formulated as shown in fig. 2.2. The relationship between the sleepers' distance in the image plane D, the real distance d of the sleepers, the focal f is explained by

$$\frac{z}{\frac{D}{2}} = \frac{f}{\frac{d}{2}} \Rightarrow \; z = \frac{fD}{d}. \tag{2.1}$$

If we take the depth value z of each rail sleeper as shown in fig. 2.2, and apply the formula we will have :



(a) (b)

FIGURE 2.2 – Scene parameters used to compute camera focal length shown on top view (a) and the image plane (b) : the distance between two railroad ties Δz, rail road ties value D, railroad tie values in the image plane $d_0 \dots d_n$ and railroad depth values $z_0 \dots z_n$.

$$\begin{aligned} z_1 = \frac{fD}{d_1}, \quad z_2 = \frac{fD}{d_2}, \\ z_1 - z_2 = \Delta z, \quad \frac{fD}{d_1} - \frac{fD}{d_2} = \Delta z, \\ f(\frac{D}{d_1} - \frac{D}{d_2}) = \Delta z, \quad f = \frac{\Delta z}{\frac{D}{d_1} - \frac{D}{d_2}}. \end{aligned} \tag{2.2}$$

If we consider n railroad sleepers as shown in fig. 2.2 the problem will then be formulated as follows :

$$
\begin{aligned}
z_0 = \frac{fD}{d_0}, \quad z_1 = \frac{fD}{d_1}, \quad z_2 = \frac{fD}{d_2}, \quad \ldots, \quad z_n = \frac{fD}{d_n} \\
f\frac{D}{d_0} = f\frac{D}{d_1} - \Delta z \quad = f\frac{D}{d_2} - 2\Delta z \quad = f\frac{D}{d_3} - 3\Delta z \quad \ldots \quad = f\frac{D}{d_n} - n\Delta z \\
n(f\frac{D}{d_0}) = (f\frac{D}{d_1} - \Delta z) + (f\frac{D}{d_2} - 2\Delta z) + (f\frac{D}{d_3} - 3\Delta z) + \ldots + (f\frac{D}{d_n} - n\Delta z) \\
n(f\frac{D}{d_0}) = fD(\frac{1}{d_1} + \frac{1}{d_2} + \ldots + \frac{1}{d_n}) - \Delta z(1 + 2 + \ldots + n) \\
n(f\frac{D}{d_0}) = fD\sum_{i=1}^{n}\frac{1}{d_i} - \Delta z(n(n+1)/2) \\
f = \frac{-\Delta z\frac{n(n+1)}{2}}{(\frac{n}{d0}) - \sum_{i=1}^{n}\frac{1}{d_i}}.
\end{aligned}
\tag{2.3}
$$

To compute f for each selected frame, we need railroad ties values in the image plane. Railroad ties in the plane image are computed from manually annotated frames as shown in fig. 2.3, n in eq. 2.3 shows the number of annotated sleepers. A minimum of 2 rails should be annotated to compute the distance between two railroad ties in the image plane. In most of the experiments 4, 6 or 8 rails are manually annotated. Note that, since the camera is assumed to be static throughout the entire sequence, this manual intervention needs only to be done in a single image for each sequence.

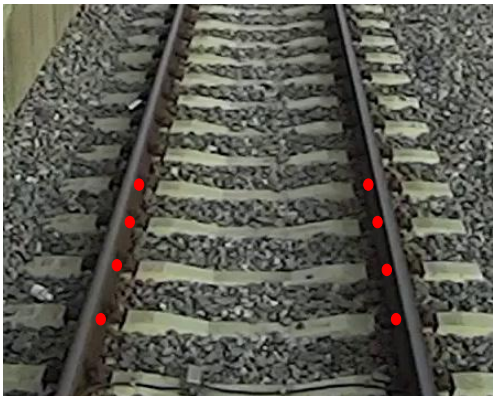


FIGURE 2.3 – Annotated railroad ties, annotation is done from left to right and starting from the inside faces of the rails.

2.3.3 Rails detection

Before computing the depth map we need to detect rails road track. For this, we used Hough transform [9] to detect the rails as long lines in each frame as seen in fig. 2.4, at which the depth map later on in computed.

FIGURE 2.4 – Example of rail detection process on one of the frames in one sequence. rail detection is processed using Hough transform on a masked region (rails region).

The rails are centred in the middle of the frame, in order to detect the long lines of rails only, we used a mask which excludes the whole pixels from the long line detection algorithm aside from the rails.

After detecting rail tracks, we label the rail that is detected in the first half of the frame, according to the width, as left and the rail that is in the second as right as shown in fig. 2.5.

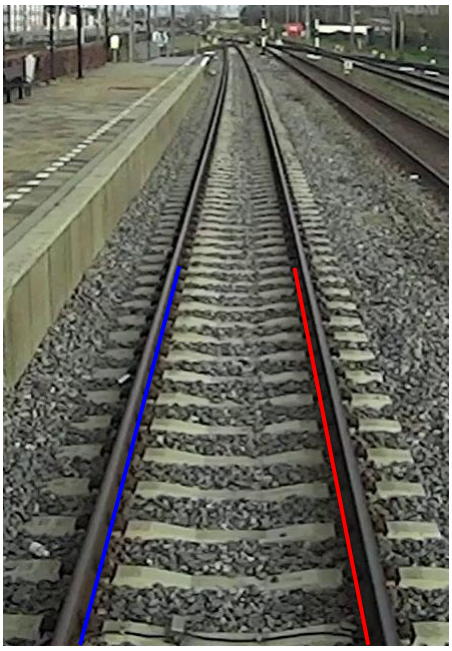


FIGURE 2.5 – Define left and right tracks in the rails : to compute the depth map inside the rail, similarly to annotating railroad ties, we defined the left and the right rail, marked respectively with blue and red.

2.4 Estimating the depth from rails

Rails depth map of a given annotated location inside the rails starting form the inside faces of the rails are computed using the average focal length with the following formula, $z = \frac{fD}{d}$. Note that this depth map extraction is only applicable in rails appearing as straight line in the image planes, we cannot generate the depth map while the train is turning.

With the definition of the left and right rail we now can compute the distance d of each pixel in the image plane from the right with its corresponding pixel from the left. Fig. 2.6 shows the corresponding depth map of the annotated frame in fig. 2.3.

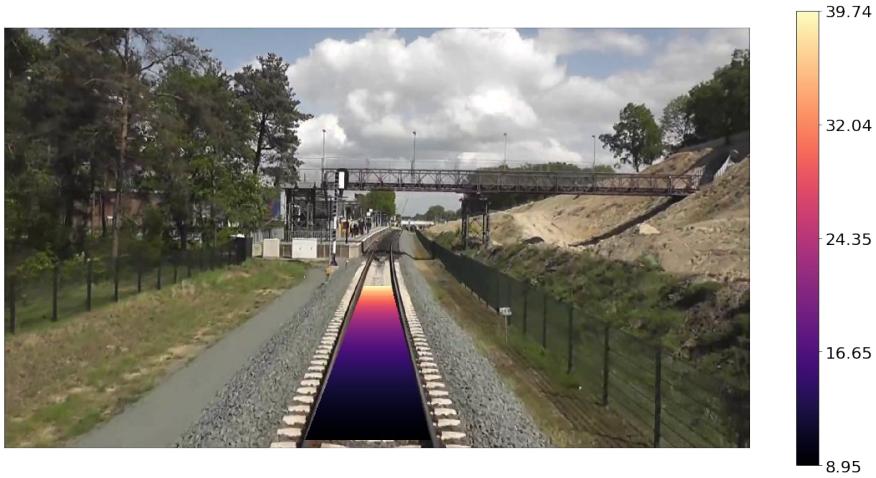


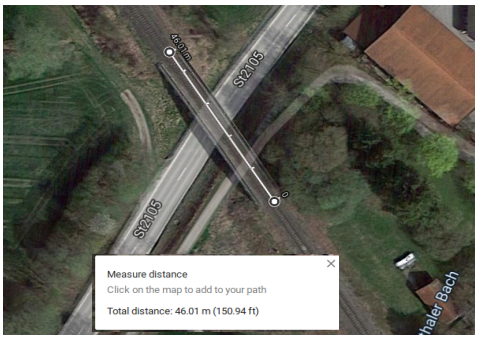
FIGURE 2.6 – Visualize depth map inside rails : the depth map corresponds to the same length of annotated left and right rail, the depth is coded here in the picture from black (nearest value of depth) to white (farthest value of depth), the color bar shows the values of depth in meters. The values of the ground truth depth are real values and not relative.

2.4.1 Validation

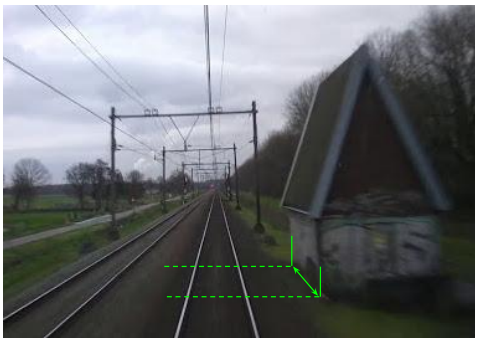
To validate the recovered depth map using our proposed technique, first we manually identified recognizable landmarks in the sequence, we computed the depth of its furthest point and closest point using the rails as reference since we only can compute the depth of the rails, then we computed its distance by subtracting the depth of the furthest point in the object and the closest one.

Afterwards, we located the same landmark from overhead imagery obtained from google maps and measured its distance to compare it to the computed distance for validation. Fig. 2.7 show 3 examples of sequence validations. In the first example we chose to validate the sequence using the bridge distance, Overhead imagery distance is about 46 meters, using the eq. 2.1 we found that the length of the bridge is 48.8 $\pm^{4}_{2}$ meters.

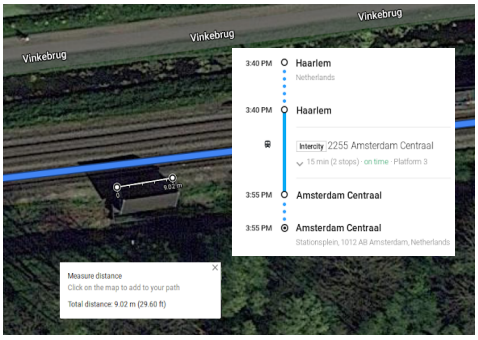
(a) Bridge length = 48.8199 $\pm^{4}_{2}$



(b) Bridge length in google maps = 46

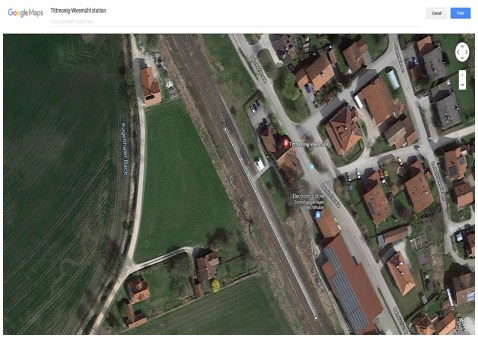


(c) Object length = 10.88 ± 2

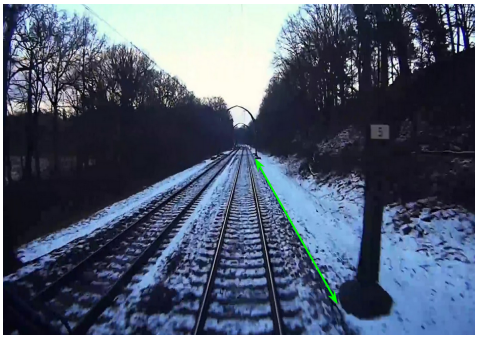


(d) Object length in google maps = 9.22

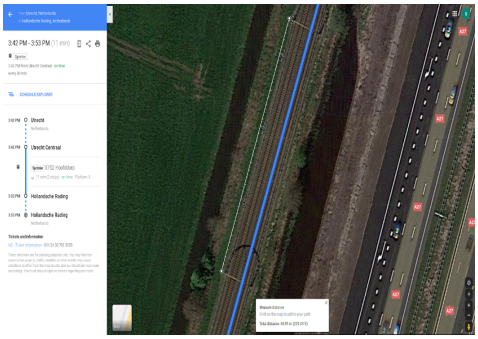
(e) Stop distance = 123.51 $\pm^{6}_{5}$ meters.



(f) Length maps = 119.85 meters.



(g) Distance marked in green = 58.65 $\pm^{3}_{2}$



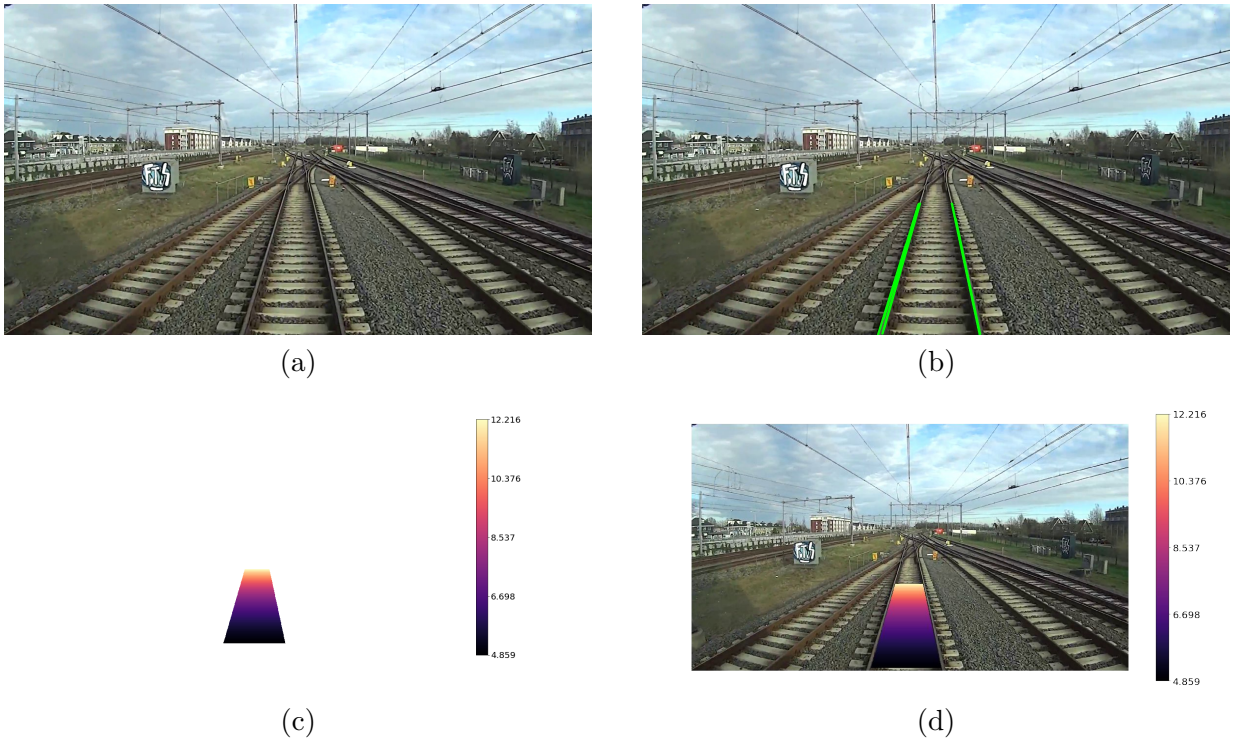
(h) Distance in google maps = 60.81

FIGURE 2.7 – Example of validation using overhead imagery (google maps in our case), for validation frames (a), (c), (e) and (g). (b), (d), (f) and (h) are the corresponding real world values in meters of the bridge, the object (small house), the stop length and the distance between two electric power traction respectively.

2.5 Data set

To generate the full dataset, we first have to compute f for each sequence in the dataset as explained in sec.2.3. Then, we detect the rails in the frames. After that, both left and right rails are detected, we finally can compute the depth map of each frame in the sequence.

An example of the whole generation process is shown in fig. 2.8.



(a) (b)

(c) (d)

FIGURE 2.8 – Example of a depth map computed from the input frame (a). depth is computed using the rails detected in (b) and left/right rails are annotated in (c). Depth (d) ranges from near (dark purple) to far (yellow) according to the color bar shown on the right. The main goal of the figure is to give an idea about the distance inside the rails, thus the density of points being smaller for from the camera does not present a problem.

2.5.1 Data set details

The data set is composed of 7 sequences with different lengths and different camera focal lengths. Fig. 2.9 shows all sequences focal lengths. The whole dataset contains 9840 frames as shown in fig. 2.10. Since the ground truth is not available for all frames, the number of depth maps available is only 6696. It is not possible to recover the rails depth map for all the frames as detecting the rails in some frames is not possible. This dataset will be used for finetuning experiments presented in chapter 3.

Video frames data set is formulated as follows :
80% of each one of the 6 sequences is used for training, 5% of each of the 6 sequences is used to separate training frames from validation frames so that validation and training frames look different. The rest (15%) is used for validation. Fig. 2.10 shows the number of frames used for each sequence.

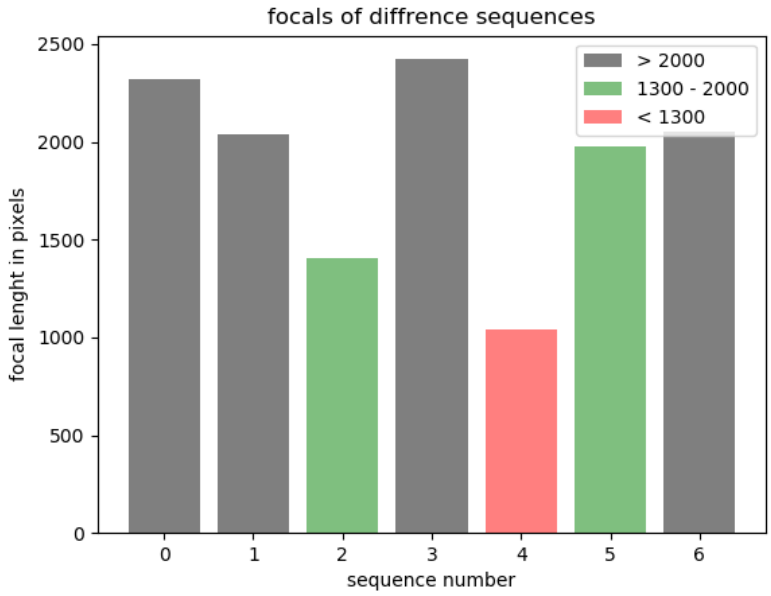


FIGURE 2.9 – Focal lengths of video sequences in the data set.

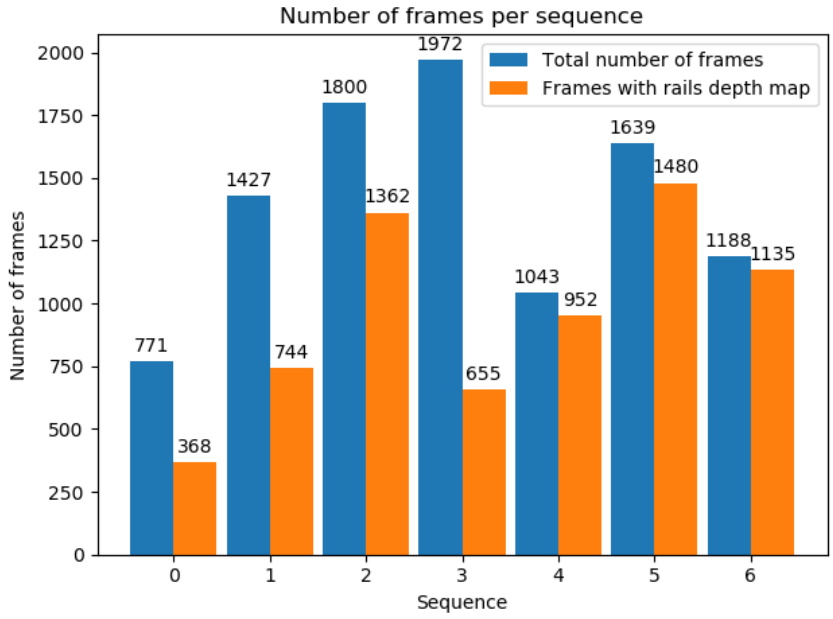
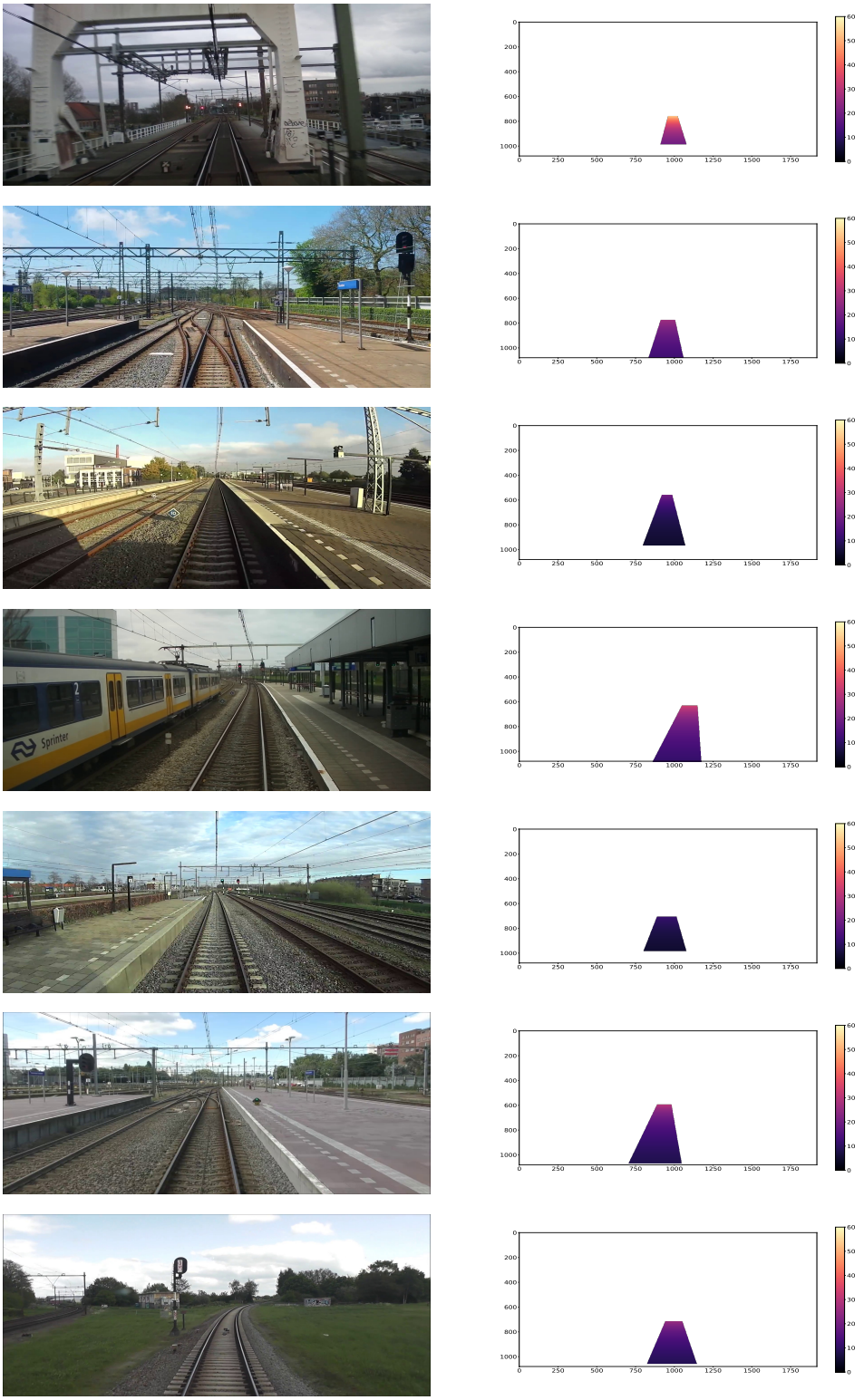


FIGURE 2.10 – Number of frames per sequence.



(a) (b)

FIGURE 2.11 – Example of input image (a) with its corresponding ground truth (b) for each sequence in the data set. Ground truth ranges from near (dark purple) to far (yellow) with a maximum set to 60 meters according to the color bar shown on the right of each depth map example.

2.6 Conclusion

In this section, we explained how we generated our train data set, and how it is possible to extract informations about the image, the focal length in our case, relying on the geometry of the scene. Having the focal length provided made it possible to compute a partially known depth values of the rails track which would serve as a supervisory signal to the network later on. The next section will cover our main approach which consists of fine tuning "Monodepth2" on train sequences, using both the focal lengths and the rail track depth values that we have computed in this section.

Chapitre 3

Finetuning Monodepth on train sequences

3.1 Introduction

In this chapter we will cover fine tuning experiments held to solve depth map estimation for train navigation.

During the first part of the chapter we will present a brief description of Monodepth2's [16] network which is actually the model we used to conduct our fine tuning experiments, its loss function, and the modification we made to the loss function during the training process. Two main sets of experiments are made. The goal of the first set is to see the influence of training the network with and without depth ground truth. The goal of the second set is to see the influence of the focal length on the finetuning process. For this purpose, different focal lengths will be used, such as kitti focal length, the sequence focal length, and an average of all sequences focal length.

3.2 Experimental methodology

In our project, we will use Monodepth2 [16] pretrained model to solve for depth estimation on train dataset. Monodepth2 [16] explained in chapter 1 has two main subnetworks : one for depth estimation and the other for pose estimation as shown in fig. 3.1.

Monodepth estimates the depth of a given image by producing a dense depth map such that it minimises the photometric reprojection error as explained in chapter 1, monodepth also uses the auto-masking method to retrieve pixels moving at the same speed as the camera as well as sky pixels from their loss function by using an identity reprojection loss on those

pixels. It is trained on the Kitti dataset [14] with Kitti focal length which is 721.5377 pixels. During our experiments, we will finetune using the same model with some changes. The first goal is to see the influence of the depth map ground truth we have computed in the rails region (see section 3.3). For this purpose, we will finetune the model first without changing anything. Then, we will exclude the rails region from the automasking technique, and finally we will include the depth ground truth we have calculated in the input of the network. For this reason, we will change the loss function of the rails to an L2 loss between the predicted output and the depth ground truth. The purpose of the second set of experiments is to see the impact of the focal length on the results. The original model is trained using Kitti focal length. In addition, we will train using the sequences focal lengths or the average focal length of all sequences computed as explained in chapter 2. During these experiments we will first use only one sequence from our dataset to train on, this sequence is chosen randomly. The chosen sequence is sequence 4 which has 1043 frames in total, 952 of them have a corresponding ground truth. Its focal length in pixels is 1039.78.

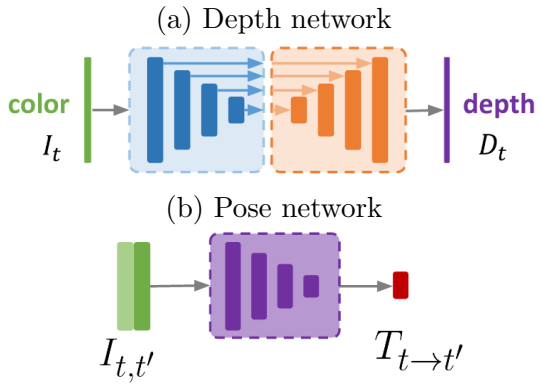


FIGURE 3.1 – Overview of both sub-networks of "Monodeth2" [16], The first sub-network is a full conv Unet model presented as the depth network in (a). The second is a separate pose estimation network (b) used to estimate the pose between two temporal frames given as input.

3.3 Impact of ground truth depth

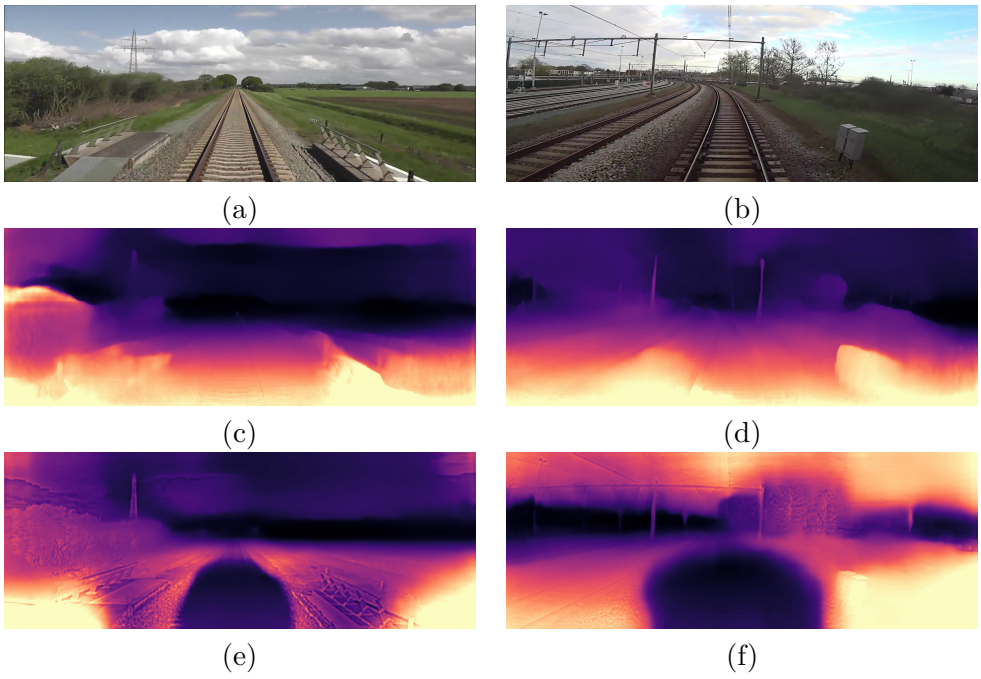
To see the impact of the ground truth on the finetuning process, we conducted the following sets of experiments. First we finetuned the model without depth ground truth, then we just excluded the rails region from the automasking loss technique, last we finetuned the model using the depth ground truth of the rails region.

After examining the influence of ground truth we proposed to improve the results in the sky region by segmenting the sky's area in order to exclude it from the automasking loss. All experiments here are conducted using the Kitti focal length, as the next section will cover the influence of the focal length of the camera. The first part of the section will cover the

experiments of training only on one sequence. We will later train the network on four sequences (see sec. 3.3.5).

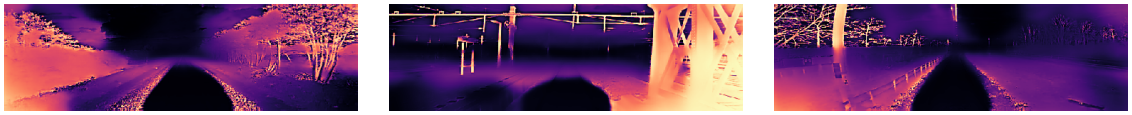
3.3.1 Fine tuning without rails depth ground truth

To solve the problem of standing objects or rails in front of the camera as shown fig. 3.2 (c) and (d) we finetuned Monodepth2 model on our train sequences dataset. During the current experiment, we finetuned the pre-trained model as it is. Just as illustrated in the qualitative results in fig. 3.2, the model creates an "infinite depth" inside the rails, depth values similar to the sky depth values, present in the fig. 3.2 as dark region inside the rails. In order to ensure that this problem is not a result of lack of data, we trained the model on the full dataset which gave the same result shown in fig. 3.3.



(a) (b) (c) (d) (e) (f)

FIGURE 3.2 – Depth results from a single input image in (a) and (b), using Monodepth2 pre-trained model in (c) and (d) and fine tuned on train data set using Kitti focal length in (e) and (f).



(a) (b) (c)

FIGURE 3.3 – Depth results from different input images using a model trained on the full set of data, to show that the problem is not a matter of less data.

3.3.2 Finetuning without masking out the rails area from the loss function

In order to avoid objects that have the same speed as the camera and stationary scenes from affecting the loss function, Monodepth2 used the auto-masking technique explained in chapter 1, which eliminates pixels belonging to those objects from the loss function. Rails which do not change appearance while the camera is moving appear as objects that have the same speed as the camera, sometimes rail sleepers appear as such, just the same case as the sky. Most of the pixels in these regions are excluded from the loss function as shown in fig. 3.4. Black pixels are removed from the loss function by the network, which results in infinite depth (the same depth as the sky) predicted by the network inside the rails area as this is the best optimisation in this case for the network.

FIGURE 3.4 – Auto-masking : example of visualisation of the auto-masking process, where black pixel are removed from the loss function.

To solve this problem, during this experiment we excluded the rails region from the auto-masking process. As shown in fig. 3.5, the new pixels present in white in the tracks area are now included in the loss function.

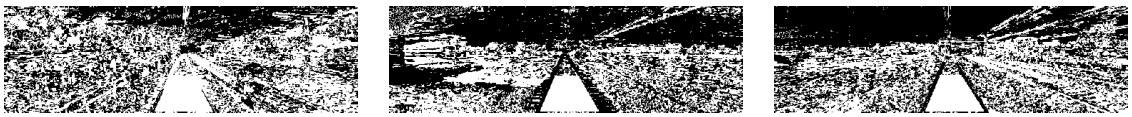


FIGURE 3.5 – Exclude rails pixels from auto-masking technique by adding a mask (white) inside the tracks. The network is now able to apply a reprojection loss on rails region. Images are taken from the same scene as fig . 3.4 but they do not represent the same frames.

As shown in fig.3.6, excluding the rails region from the auto-masking technique does not solve the problem of the rails region appearing far from the camera.

Comparing to the fine tuning without modifications results, excluding the rails from the auto-masking technique does not make the whole area of the tracks appear to have "infinite depth" (having the same depth as the sky), instead only the last part of the rails (the intersection of the two tracks) have a similar depth to the sky as shown in fig .3.6.

For this reason, we have tried to expand the white area to cover the rails, and outside the rails, or to only include the railroad ties using a dilation or an erosion as shown in fig. 3.7.

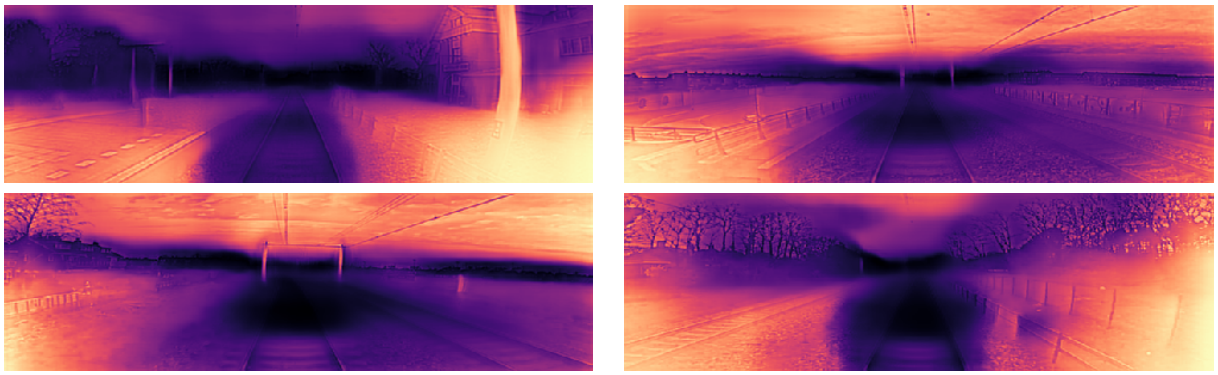


FIGURE 3.6 – Predicted depth map examples for four different frames of fine-tuning experiment without including the rails in the automasking technique. Farthest values of depth appears in dark purple, such as the sky area. The rails also appear in dark purple which means that the rails are estimated as objects located far from the camera, especially the regions around the tracks intersection.

(a) (b) (c)

FIGURE 3.7 – Auto-masking technique used with "Monodepth2" excluding the rails area, black pixels are where the loss function is not applied, white pixels are taken in consideration in the reprojection loss function. the mask in (a) and (b) dilated to cover the rails, the mask in (b) is eroded so that it wouldn't cover the whole area.

Results shown in fig.3.8 have shown a slight improvement compared to the previous one, no part of the tracks appear to be very far from the camera and have the same depth as the sky, however, the depth inside the tracks is still inappropriate as the tracks still appear farther then they should be.

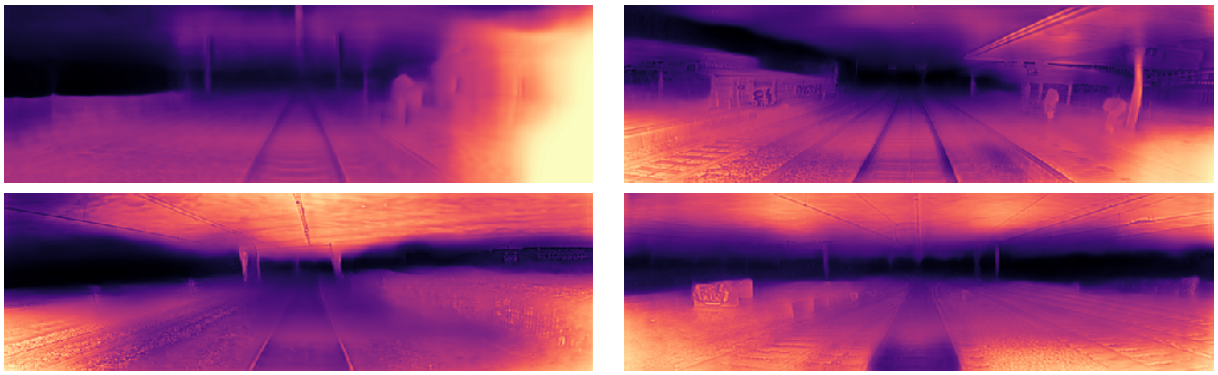


FIGURE 3.8 – Predicted depth map examples for different frames of the fine-tuning experiment without including the rails in the automasking technique. These results are the results of the fine-tuning experiment using the eroded mask as present in fig.3.7 in (c). The tracks do not appear very far as in the previous experiments, however, the depth in the rails region still appear farther than it should be.

3.3.3 Finetuning with rails depth ground truth

As applying the reprojection loss on the rails region was not sufficient to solve the problem, we have applied a different loss function for the rails area only. We use an $L2$ loss between the predicted depth output $\hat{d}$ inside the rails and the depth ground truth d over the pixels N which have the depth values provided. Note that the depth value of each pixel is in meters as shown in chap.2.

$$Y_{gt} = \frac{1}{N}\sum(d_i - \widehat{d}_i)^2 \tag{3.1}$$

As fig. 3.9 illustrates, adding the ground truth loss for the rails region has solved the "infinite depth" problem of the rails (where depth predictions are the same as the depth predictions of the sky in the scene), the depth map is also generated with values in meters. However, one other problem persists which is that the network poorly predicts the depth of the sky.

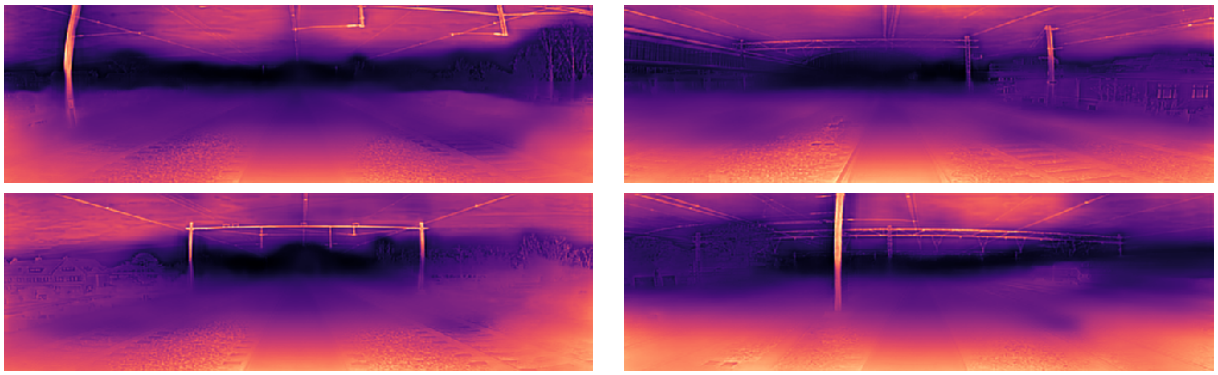
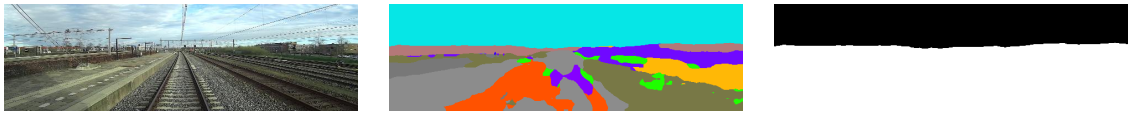


FIGURE 3.9 – Predicted depth map examples for fine tuning experiment. Here we have used the corresponding depth ground truth values.

3.3.4 Finetuning with rails depth ground truth and sky masking

To solve the sky problem, just as we excluded the rails from the auto-masking process, we also excluded the sky region by segmenting the sky and aerial lines using [41] [42] 's code. We gave the sky mask of each corresponding image as an input to the network.



(a) (b) (c)

FIGURE 3.10 – Sky and aerial lines segmentation from input image in (a), results of segmentation in (b), sky mask used as input to the network in (c).

Just as qualitative results show in fig. 3.13, segmenting the sky region has solved the apparent problem of the sky depth, although the edges are still not well defined the rails depth map show appropriate values compared to previous experiments (the depth inside the rails is not far comparing to its surroundings).

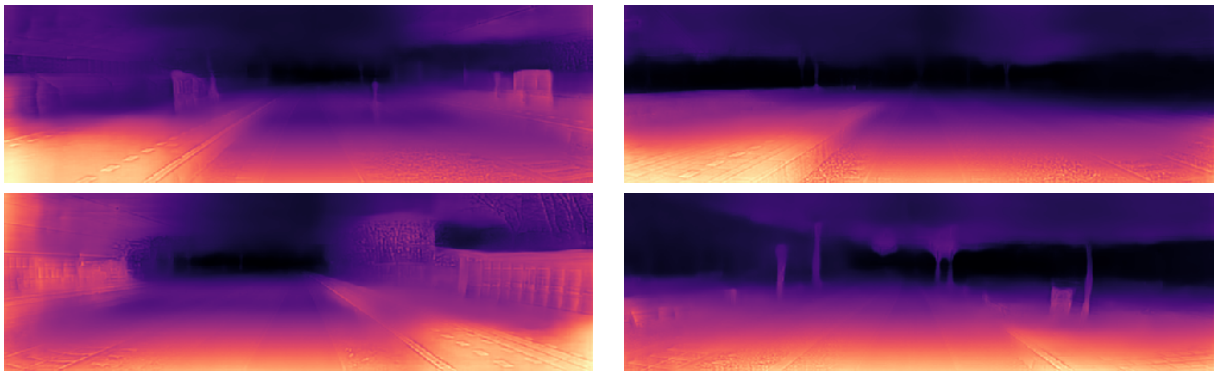


FIGURE 3.11 – Predicted depth map examples for fine tuning experiment using both ground truth depth values in the loss function and the sky segmentation mask. By providing the sky segmentation mask the network would apply the loss in the sky's area.

3.3.5 Finetuning with rails depth ground truth and sky masking on four sequences

In this experiment we used four sequences for fine-tuning with ground truth provided and the the segmented sky excluded from the auto-masking technique. Sequences are chosen randomly. Sequences used for the fine-tuning are : sequences $0, 1, 4, 7$ with corresponding focal lengths $2323.2, 2037.66, 1039.78, 2054.41$. Fig. 3.12 shows different predicted depth map examples. Including the sky and aerial lines segmentation mask have shown a noticeable improvement in the results. It helped avoid the aerial lines to appear as standing object in front of the camera as well as other depth artifacts.

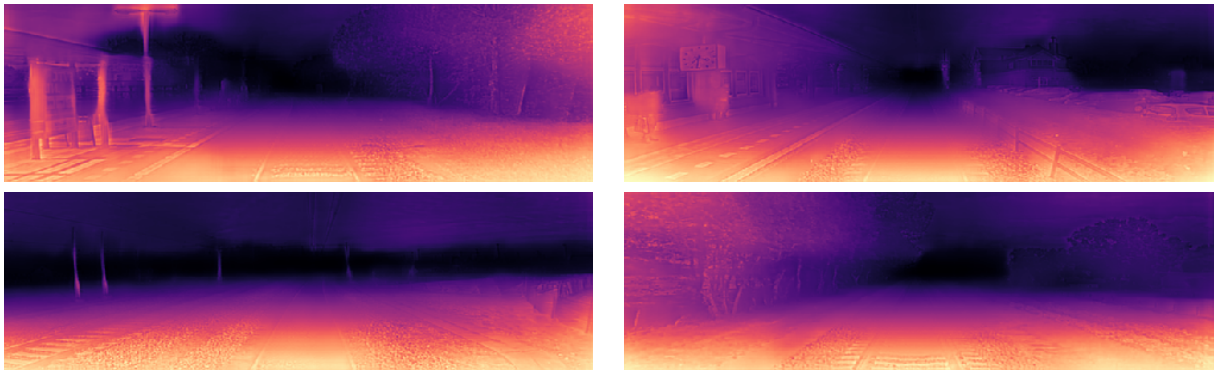


FIGURE 3.12 – Predicted depth map examples for fine tuning experiment with four sequences instead of only one sequence. Examples are chosen randomly.

3.3.6 Comparison of the different experiments

As have previous results shown, providing the ground truth depth map as a supervisory signal to the network have shown important improvement in the predicted depth maps. It has solved the problem of very far depth values present in the depth maps as well as inappropriate depth inside the rails tracks.

Qualitative results in fig. 3.13 have shown a noticeable improvement adding the sky and aerial lines segmentation process which helped avoid the aerial lines to appear as standing object in front of the camera as well as other depth artifacts.

Quantitative results to compare all previous experiments are shown in Tab. 3.1. We compare the results of several experiences finetuned differently. First without using the ground truth provided for the area of the rail (No GT), second using a mask on the area of the rail (Masked Rails), third using the ground truth map provided for the area of the rail (Provided GT), then using both the ground truth provided and a mask for the sky (GT and sky mask), and last using all four sequences for training with both rails ground truth and the sky mask (four seq exp). The results in Tab. 3.1 show that providing the network with rails ground truth increases accuracy, which is especially remarkable on the metrics, where the root mean square error (RMSE) is $8.1283m$ when fine-tuning without ground truth and $4.6496m$ when training providing the ground truth. the same goes for the Root Mean Squared Logarithmic Error (RMSE log) and the absolute relative error (Abs rel). According to the metrics, the best one-sequence experiments is when we used both the ground truth depth maps and the sky segmentation mask where the value of the RMSE metric is $4.5691m$ compared to $4.6496m$ when only the ground truth depth maps are provided. Results are best as expected when training using four sequences where the RMSE is $2.2275m$ compared to $4.5691m$ when fine-tuning using one sequence only.

	RMSE	RMSE *log*	Abs Rel
Monodepth 2	5.0217	0.5517	0.7060
No GT	8.1283	0.9674	1.1328
Masked Rails	4.7548	0.5455	0.7001
Provided GT	4.6496	0.5289	0.6868
GT and sky mask	**4.5691**	**0.5252**	**0.6822**
Four seq exp	**2.2275**	**0.36918**	**0.3898**

TABLE 3.1 – Quantitative results of the different fine tuning experiments. The last column presents the absolute relative error. Note that "No GT" presents the experiment without ground truth provided, "Masked Rails" is the experiment with the mask used on rails, "Provided GT" is the experiment where the ground truth depth is provided, "GT and sky mask" presents the experiment with both ground truth and sky segmentation provided, the last experiment "Four seq exp" presents the fine tuning using the ground truth and the sky segmentation on four sequences instead of one sequence which as expected have the best results.

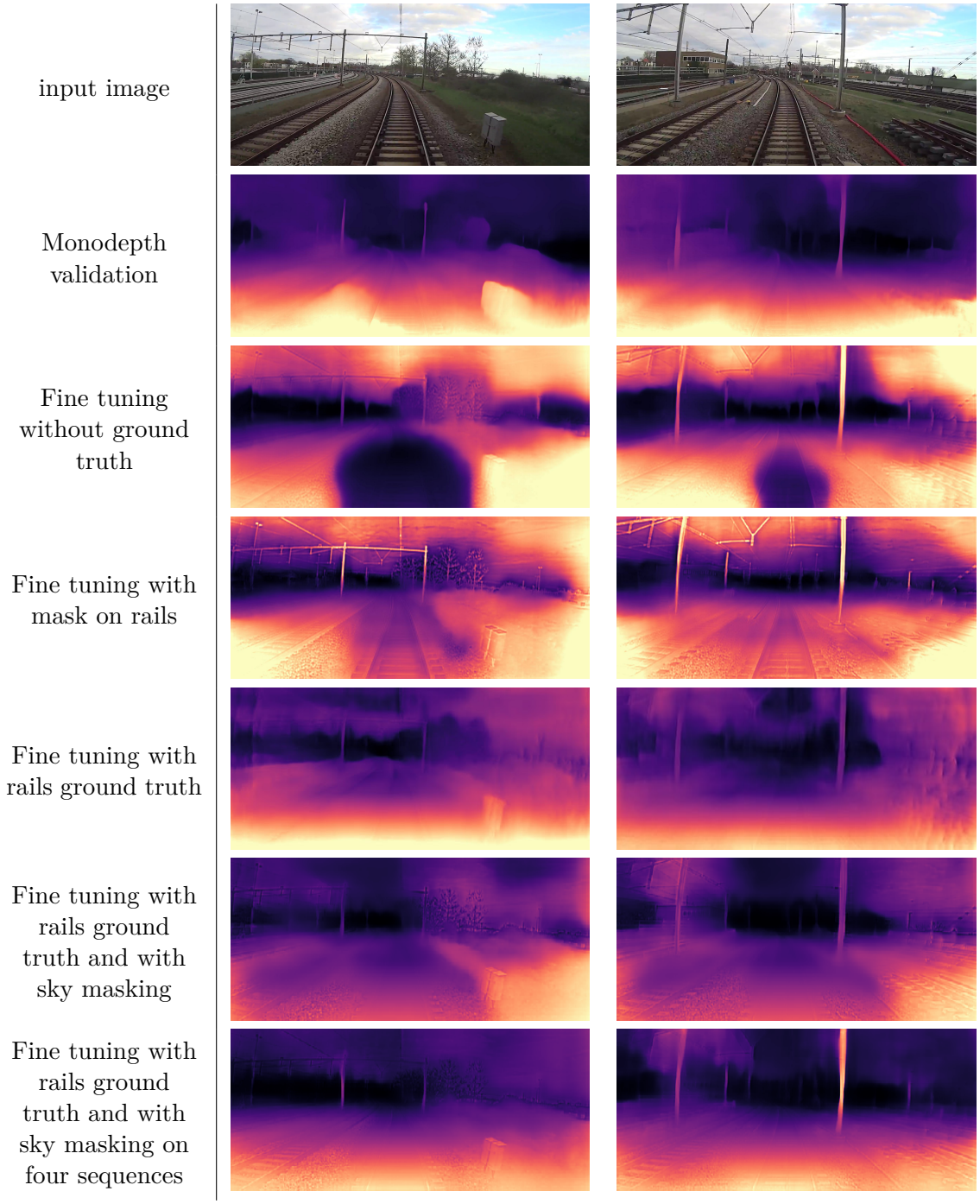


FIGURE 3.13 – Qualitative results of different experiments on two input images. Depth map varies from yellow (closest) to dark blue (farthest). We can see that the model with all conditions applied (depth ground truth and sky and aerial lines masking) enhanced the depth map of the ground and its surroundings and omitted depth artifacts from the sky.)

3.4 Influence of focal length

The main goal of this set of experiments is to see the influence of the focal length on the resulting depth map. To do so, we will be fine-tuning the network on one sequence only with a specific focal length. We first conducted two experiments on a single, randomly-selected sequence : 1) Fine tuning using the Kitti focal length. 2) fine tuning using the sequence focal length. Then, we experimented on four sequences, trying first with the Kitti focal length then each sequence's focal length and last we fine-tuned on the average of all sequences' focal lengths.

3.4.1 Fine tuning on one sequence

Fine-tuning using the Kitti focal length

The pretrained model is trained initially using the Kitti focal length which has a value of 721.5377. During the following experiment we will fine tune our model with the same focal length.

As explained in chapter 2, the ground truth loss is only applied on the rails area, where the ground truth depth values in meters are provided. The reprojection loss is applied outside the rails area. Fig. 3.14 shows a different validation example during different epochs for one input image. Fig. 3.15 shows other examples for different input images after training the network.

As shown in fig. 3.14 depth map results have improved, although we are using the same focal length for all the sequences which are actually captured using different cameras with different focal lengths.

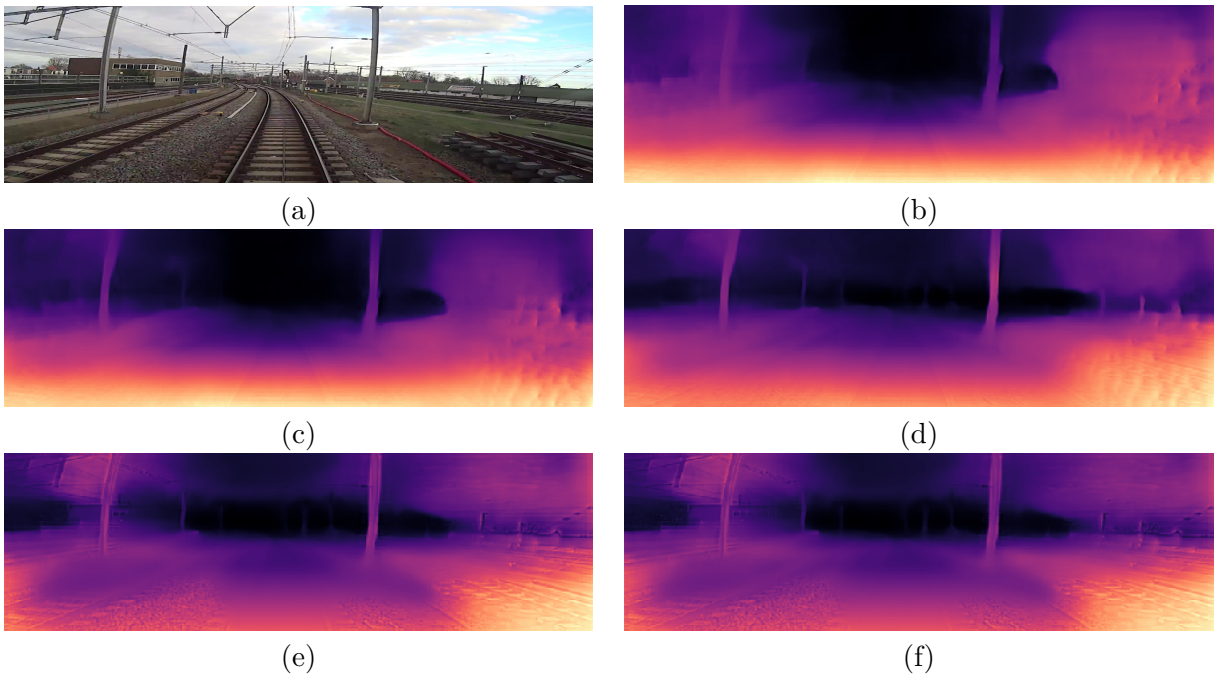


FIGURE 3.14 – Validation examples : depth maps from a single input image in (a) during validation process in the first epoch in (b), second epoch in (c), fourth epoch in (d), epoch 25 in (e) and epoch 30 in (f).

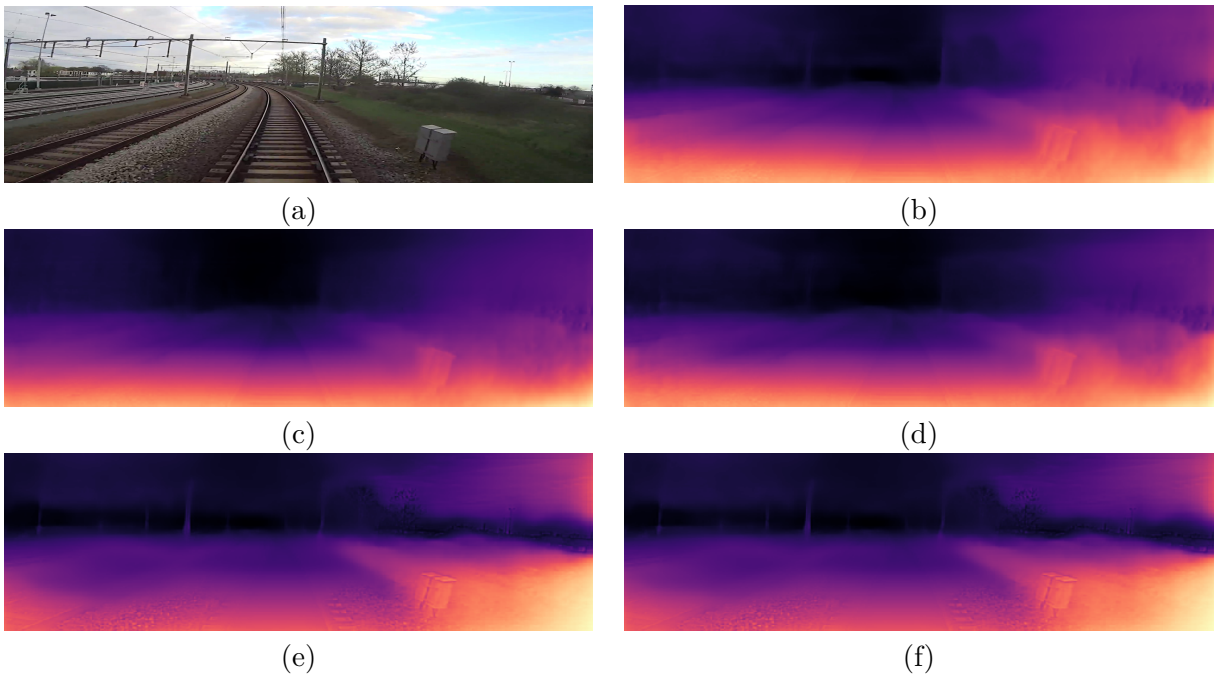
FIGURE 3.15 – Other examples of fine-tuning using the Kitti focal length for different input images.

Fine-tuning using sequence focal length

In this experiment we have changed the focal length during the finetuning process, so the model which actually is trained using Kitti focal length will be finetuned on one sequence using the sequence focal length which was estimate using the methodology presented in chapter 2.

Fig. 3.16 shows depth maps predicted during the validation stages of some epochs for the same input image. Fig. 3.16 shows depth maps estimation for different other input images after fine-tuning.

As shown in fig. 3.16 depth map results haven't shown a significant qualitative difference compared to the fine-tuning experiment using the Kitti focal length. Both fine-tuning experiments have shown a significant improvement for depth map estimation results.



(a) (b) (c) (d) (e) (f)

FIGURE 3.16 – Validation examples : depth map from a single input image in (a) during validation process of fine tuning using sequence focal of the first epoch in (b), second epoch in (c), fourth epoch in (d), epoch 25 in (e) and epoch 30 (f).

FIGURE 3.17 – Other examples for finetuning using the sequence focal length for different input images.

3.4.2 Fine tuning on four sequences

Fine-tuning using Kitti focal length

In this experiment we used four sequences for finetuning with ground truth provided and the sky and the aerial lines of the train segmented. Fig.3.18 shows different predicted depth map examples.

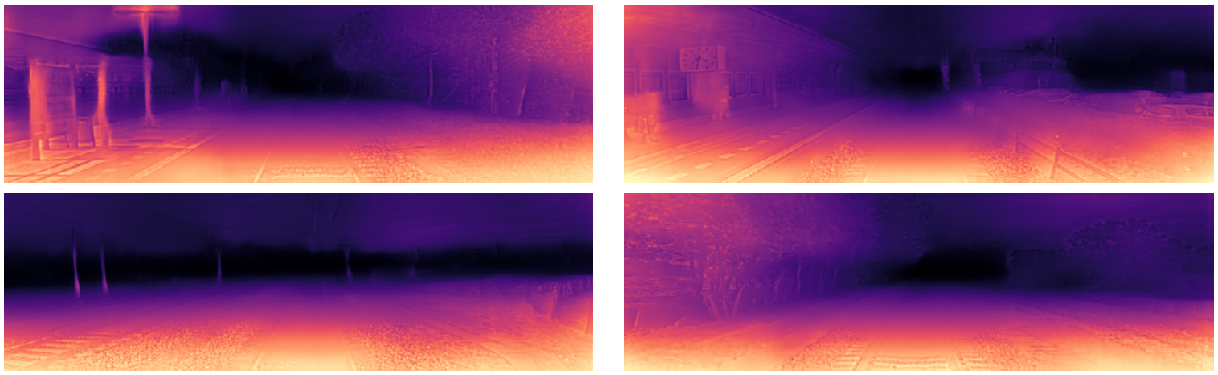


FIGURE 3.18 – Predicted depth map examples for fine tuning experiment with the Kitti focal length on four sequences.

Training with four sequences with the Kitti focal length has shown enhanced depth map estimation than training only with one sequence notably far in the scene by the end of the rails where depth in previous experiments seem farther than what it should be. The depth map is also less fuzzy and details of the scene can be better observed.

Fine-tuning using each sequence focal length

During this experiment, we used each sequence focal length during fine tuning.

Quantitative results shown in fig. 3.19 shows worse results than the previous experiment. Using a different focal length for each sequence during the traning may have confused the network and made it overfit on each ground truth provided for the rails. Being specific about the focal length may have caused the network a lack of generalisation.

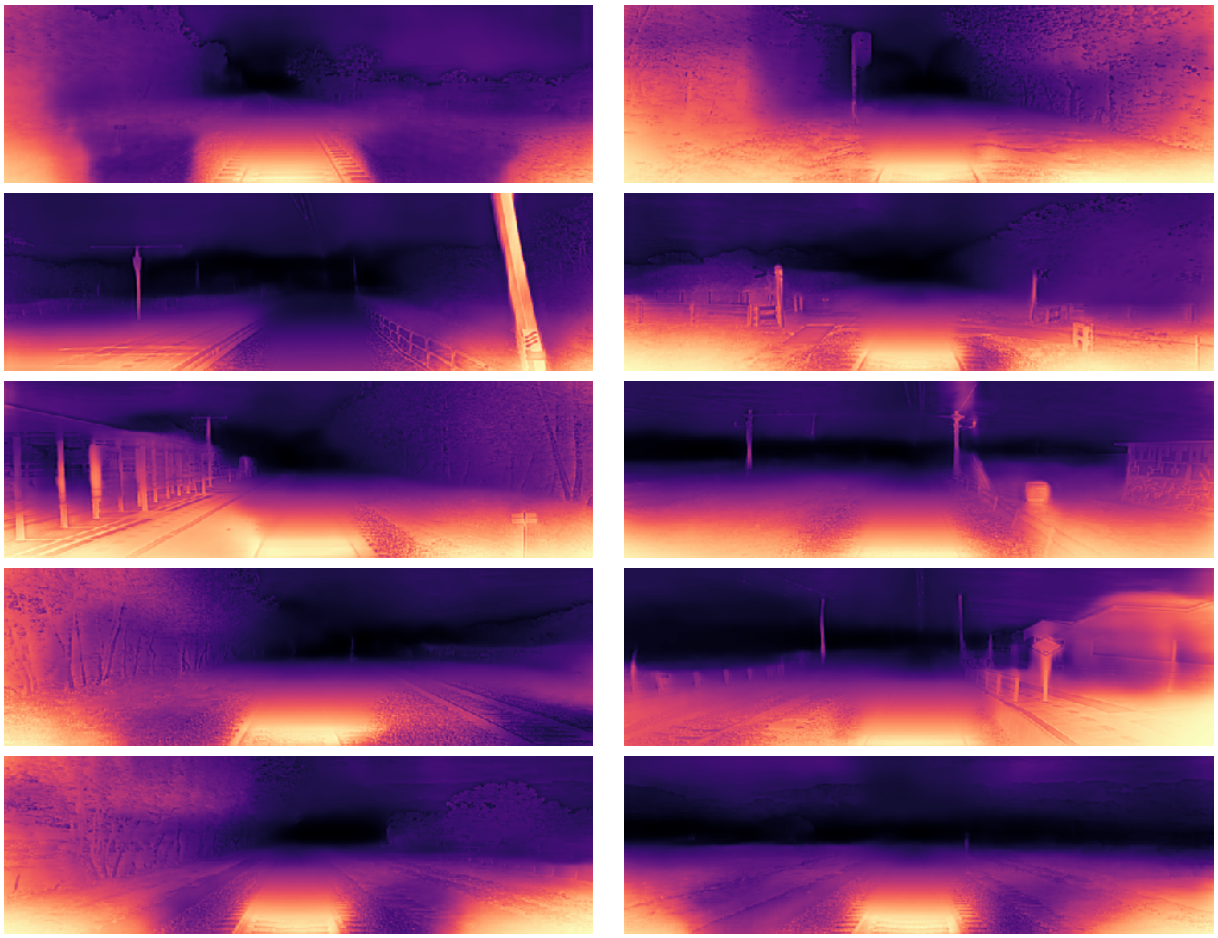


FIGURE 3.19 – Depth map prediction examples for fine-tuning experiment using each sequence focal length. Most of the results show inappropriate depth values either inside the tracks or in the left and right areas of the tracks.

Fine tuning using average focal length

In this experiment we used the average focal length of all sequences in the dataset during training, As shown in fig. 3.20, depth map inside the rails have improved along with the image details compared to the previous experiment where we fine tuned the model using four different focal lengths.

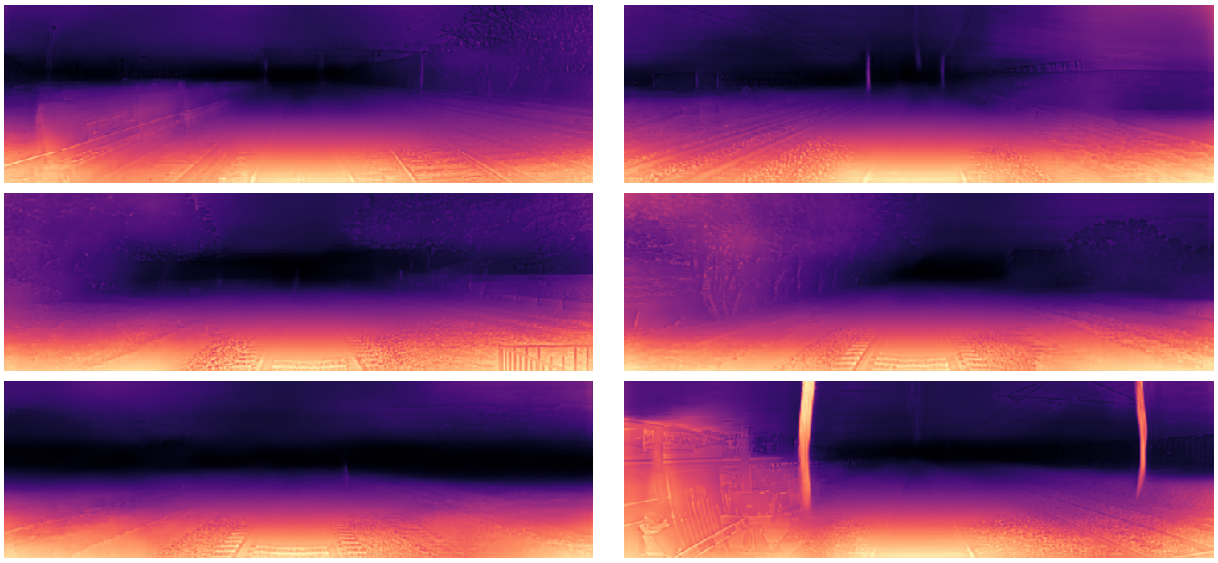


FIGURE 3.20 – Depth map estimation examples of the fine tuning experiment using the average focal lengths of all sequences used in the fine tuning process.

Fine-tuning with four sequences using the Kitti focal length and using an average focal length have shown similar results, where depth map predictions of the scenes are more detailed and better than the fine-tuning experiments with only one sequence.

3.4.3 Results comparison

The reason for these sets of experiments is to see the influence of the focal length alone. All the experiments are trained using ground truth loss in the rails region, while not all the frames have the ground truth provided, a lot of them do as explained in chap. 2. Frames who have the ground truth provided, an $L2$ loss function is applied in the rails area, while for the others for which the ground truth is not provided, the reprojection loss is applied in the area.

All the frames have the sky and the aerial lines of the train segmented and excluded from the auto-masking process. The only change made in the experiments is the focal length used.

The qualitative results show no large difference in the predicted depth maps of different experiments, apart from the sky's region which appears more fuzzy using the Kitti focal length for fine tuning. Table 3.2 shows quantitative results of different experiments. Fig. 3.21 shows qualitative results of all different experiments for the same input image.

	RMSE	RMSE *log*	Abs Rel
Monodepth 2	5.0217	0.5517	0.7060
1 Kitti FL	4.5691	0.5252	0.6822
1 Sequence FL	4.6521	0.5286	0.6743
4 Kitti FL	2.2275	0.36918	0.3898
4 Sequence FL	2.2979	0.3741	0.3965
4 average FL	**2.2408**	**0.3714**	**0.3915**

TABLE 3.2 – Quantitative results. Comparison of fine tuning experiments using different focal lengths. All fine tuning experiments in this table are conducted with both the ground truth depth maps and the sky segmentation provided. "1 Kitti FL" is the experiment using one sequence only with kitti focal length during finetuning, "1 Sequence FL" is the experiment with one sequence and finetuning using the sequence focal length. "4 Kitti FL" is the experiment of finetuning using all 4 sequences and using Kitti focal length, "4 Sequence FL" is the experiment of finetuning all four sequences using each sequence focal length and last "4 average FL" is the experiment using an average focal length of all the sequences used in the training.

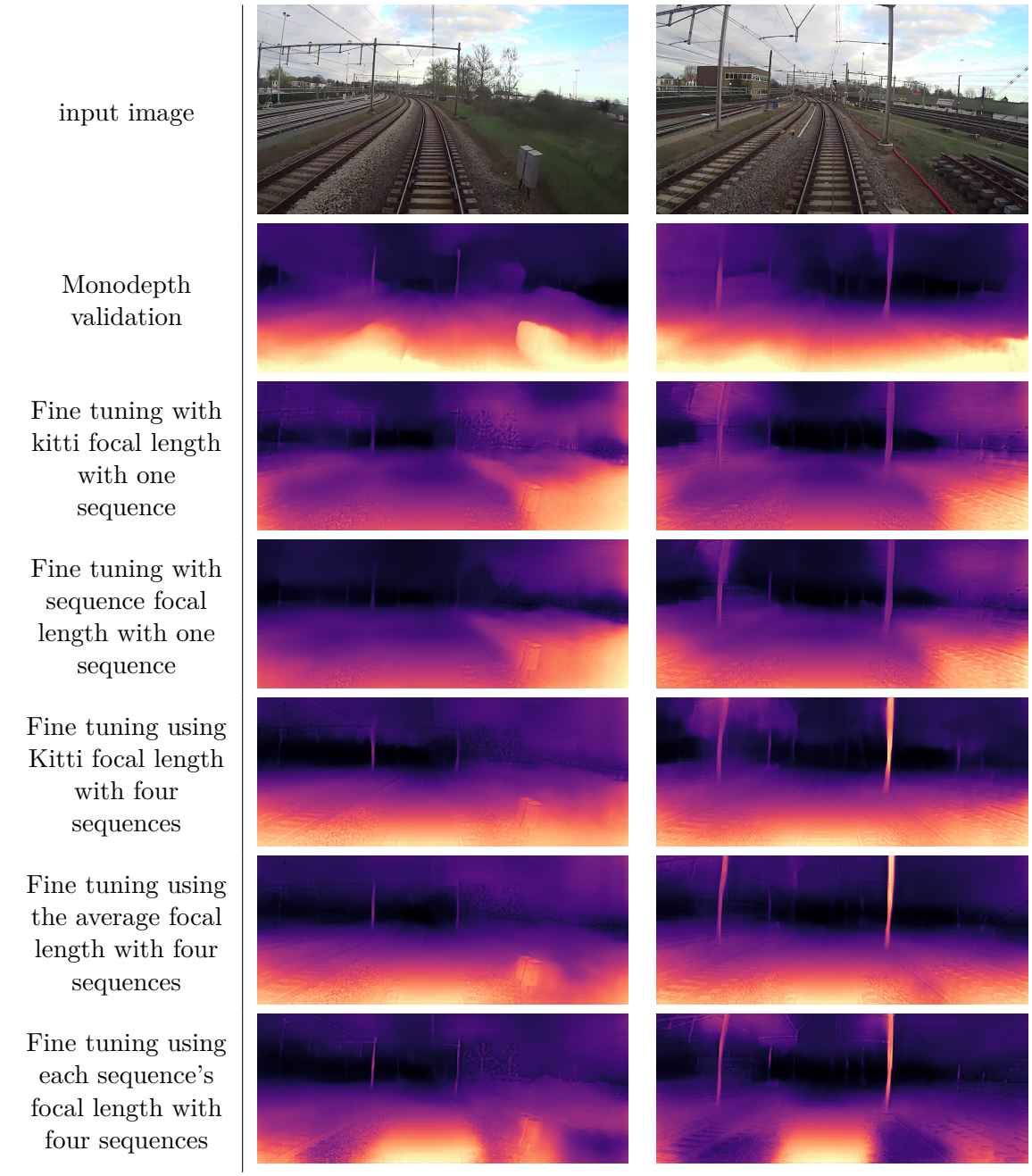


FIGURE 3.21 – Qualitative results of different experiments on two input images, depth maps as mentioned before varies from yellow (closest) to dark blue (farthest). All experiments have both rails ground truth depth maps and sky segmentation provided. Monodepth2 depth maps on those two frames are also provided for the purpose of comparison.

3.5 Failure cases examples

Failure cases examples show that there still are some frames where the corresponding depth map predictions show that the depth inside the rails is farther than its surroundings as shown in fig. 3.22. This may because of the difference of the focal length of each sequence as these failures are only present in the fine tuning experiments using more than two sequences. Fine tuning using only one sequence did not show such failures. Another failure case shown here is the depth of the rails adjacent to the main rails, which appears farther than it should be, this problem may be solved if we also provide the rails depth map of the adjacent rails

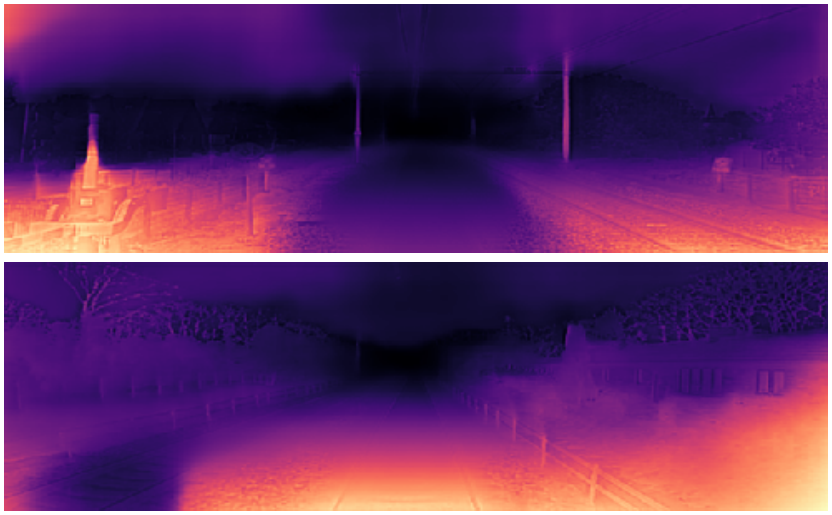


FIGURE 3.22 – Example of failure cases. All experiments show inappropriate predictions of depth map values inside the main rails or in the adjacent rails.

3.5.1 Conclusion

During the finetuning experiments, adding the ground truth depth maps of the rails as a supervisory signal to the network has shown a significant improvement in the results, the same as excluding the sky's region from the from the automasking technique applied on the loss function. As for the focal length, using an average focal length was a good choice for the network, using however multiple choices for the focal length would confuse the network during training.

Conclusion

Depth is a key component for different applications as it brings important informations about the scene, autonomous driving and navigation being one of the most popular ones. Unfortunately, not all domains have gained the same attention. For instance we don't find work for single image depth estimation designed specifically for trains. We also find more important contributions in car datasets compared to train datasets.

In this work, we were interested in the estimation of depth maps from a single RGB image for train Sequences. To this end, we first introduced a train (subway) dataset. The dataset includes partially known depth maps for corresponding sequences used as a supervisory signal for a network previously trained on a different dataset. To extract the ground truth depth map, we first chose different train navigation sequences. For each one of the sequences, we started by determining the focal length using rails standard parameters and geometry constraints.

After computing the focal length, we validated each sequence using overhead imagery by choosing an object in the scene, computing its size using the calculated focal length and comparing it to the distance provided by the overhead imagery. After validating the focal length, we detected the rails track for the sequences, then we computed the corresponding depth map of the rails for each corresponding frame. We used these depth maps as a supervisory signal to the network. We finetuned the Monodepth2 model [16] using the generated dataset. We have conducted different experiments to see the influence of the retrieved ground truth depth maps on the fine-tuning process, along with the influence of the focal lengths of the sequences.

First, we finetuned the network without the ground truth depth maps provided. Without using the ground truth depth, the network was not able to estimate the depth inside the rails, the estimated depth inside the tracks was very far, similar to the depth of the sky. By providing the network with the ground truth associated with the frames of each sequence, the problem of the depth inside the tracks have been solved. To enhance the depth of the sky and the aerial lines, we have provided segmentation of the sky for each frame to the network which has improved the results significantly. The goal of the second set of experiments was to see the influence of the focal length. First, we trained the network with the Kitti focal length, then using the sequence focal length, and last using an average focal length of all sequences used during training. Fine-tuning experiments with the Kitti focal length and the

average focal length have shown close results. However, providing the network with different focal lengths for each sequence during training has shown a significant problem in the depth estimation, where a small part of the rail tracks seems farther than inside the rail tracks. The influence of the focal length estimation error however will take part in a future work.

We also make an important observation on the number of sequences used for training. While there is a remarkable difference between using only one sequence and four sequences, where an important improvement is brought by training with four sequences, we have noticed that training with more than four sequences has its own problems. Indeed, training with more than four videos makes the network aware of the tracks adjacent to the main track, for which we do not have the ground truth depth maps provided. For this reason, inside these rails the network predicts very far depth, similar to the depth predicted for the sky, just the same issue we had encountered when we fine-tuned without the ground truth provided for the main tracks. Although this solution has solved the problem of trains depth maps, there still are a lot of improvement that could be added to the project. We can provide more corresponding depth maps as the network works best for sequences who have more depth map provided. W can also provide the depth of the adjacent tracks so it would be possible to train using a bigger dataset. We can solve the depth problem for when the train is inside the tunnel, and train with more varied weather conditions.

Bibliographie

[1] A. Atapour-Abarghouei et T.P. Breckon : Real-time monocular depth estimation using synthetic data with domain adaptation. *In Proc. Computer Vision and Pattern Recognition*, pages 1–8. IEEE, June 2018.

[2] Google AI Blog : https://ai.googleblog.com/2019/12/improvements-to-portrait-mode-on-google.html.

[3] Yohann Cabon, Naila Murray et Martin Humenberger : Virtual kitti 2, 2020.

[4] Zhao Chen, Vijay Badrinarayanan, Gilad Drozdov et Andrew Rabinovich : Estimating depth from rgb and sparse sensing. *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2018.

[5] Marius Cordts, Mohamed Omran, Sebastian Ramos, Timo Rehfeld, Markus Enzweiler, Rodrigo Benenson, Uwe Franke, Stefan Roth et Bernt Schiele : The cityscapes dataset for semantic urban scene understanding. *In Proc. of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016.

[6] Angela Dai, Angel X. Chang, Manolis Savva, Maciej Halber, Thomas Funkhouser et Matthias Niessner : Scannet : Richly-annotated 3d reconstructions of indoor scenes. *In Proc. Computer Vision and Pattern Recognition (CVPR), IEEE*, 2017.

[7] Scharstein Daniel, Szeliski Richard et Zabih Ramin : A taxonomy and evaluation of dense two-frame stereo correspondence algorithms. *In International Journal of Computer Vision*, 2002.

[8] Ruofei Du, Eric Turner, Maksym Dzitsiuk, Luca Prasso, Ivo Duarte, Jason Dourgarian, Joao Afonso, Jose Pascoal, Josh Gladstone, Nuno Cruces, Shahram Izadi, Adarsh Kowdle, Konstantine Tsotsos et David Kim : DepthLab : Real-time 3D Interaction with Depth Maps for Mobile Augmented Reality. *In Proceedings of the 33rd Annual ACM Symposium on User Interface Software and Technology*, UIST. ACM, 2020.

[9] Richard O Duda et Peter E Hart : Use of the hough transformation to detect lines and curves in pictures. *Communications of the ACM*, 15(1):11–15, 1972.

[10] NAVER LABS Europe : *VKitti*. https://europe.naverlabs.com/blog/announcing-virtual-kitti-2/.

[11] Huan Fu, Mingming Gong, Chaohui Wang, Kayhan Batmanghelich et Dacheng Tao : Deep Ordinal Regression Network for Monocular Depth Estimation. *In IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2018.

[12] A Gaidon, Q Wang, Y Cabon et E Vig : Virtual worlds as proxy for multi-object tracking analysis. *In CVPR*, 2016.

[13] Ravi Garg, BG Vijay Kumar, Gustavo Carneiro et Ian Reid : Unsupervised cnn for single view depth estimation : Geometry to the rescue. *In European Conference on Computer Vision*, pages 740–756. Springer, 2016.

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

[15] Clément Godard, Oisin Mac Aodha et Gabriel J. Brostow : Unsupervised monocular depth estimation with left-right consistency. *In CVPR*, 2017.

[16] Clément Godard, Oisin Mac Aodha, Michael Firman et Gabriel J. Brostow : Digging into self-supervised monocular depth prediction. *The International Conference on Computer Vision (ICCV)*, October 2019.

[17] Zhao Hang, Gallo Orazio, Frosio Iuri et Kautz Jan : Loss functions for image restoration with neuralnetworks. *In IEEE Transactions on Computational Imaging*, December 2016.

[18] Katrin Honauer, Ole Johannsen, Daniel Kondermann et Bastian Goldluecke : A dataset and evaluation methodology for depth estimation on 4d light fields. *In Asian Conference on Computer Vision*. Springer, 2016.

[19] Junjie Hu, Mete Ozay, Yan Zhang et Takayuki Okatani : Revisiting single image depth estimation : Toward higher resolution maps with accurate object boundaries. *In 2019 IEEE Winter Conference on Applications of Computer Vision (WACV)*, 2019.

[20] Kudan inc. : *Depth cameras and RGB-D SLAM*. https://www.kudan.io/jp/archives/526.

[21] Iro Laina, Christian Rupprecht, Vasileios Belagiannis, Federico Tombari et Nassir Navab : Deeper depth prediction with fully convolutional residual networks. *In 3D Vision (3DV), 2016 Fourth International Conference on*, pages 239–248. IEEE, 2016.

[22] Wenbin Li, Sajad Saeedi, John McCormac, Ronald Clark, Dimos Tzoumanikas, Qing Ye, Yuzhong Huang, Rui Tang et Stefan Leutenegger : Interiornet : Mega-scale multi-sensor photo-realistic indoor scenes dataset. *In British Machine Vision Conference (BMVC)*, 2018.

[23] Fangchang Ma et Sertac Karaman : Sparse-to-dense : Depth prediction from sparse depth samples and a single image. *ICRA*, 2018.

[24] Wrenninge Magnus et Jonas Unger : Synscapes : A photorealistic synthetic dataset for street scene parsing, 2018.

[25] Corey Montella, Timothy Perkins, J. Spletzer et M. Sands : To the bookstore! autonomous wheelchair navigation in an urban environment. *In FSR*, 2012.

[26] Pushmeet Kohli Nathan Silberman, Derek Hoiem et Rob Fergus : Indoor segmentation and support inference from rgbd images. *In ECCV*, 2012.

[27] Karlsruhe Institute of Technology : *Kitti*. http://www.cvlibs.net/datasets/kitti/eval_depth.php?benchmark=depth_prediction.

[28] Yvon Rozijn : *railcabrides*. https://railcabrides.com/en/.

[29] Nathan Silberman : *NYU*. https://cs.nyu.edu/~silberman/datasets/nyu_depth_v2.html.

[30] Cityscapes Team : *Cityscapes website*. https://www.cityscapes-dataset.com/.

[31] Shao uanjie, Li Lerenhan, Ren Wenqi, Gao Changxin et Sang Nong : Domain adaptation for image dehazing. *In Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2020.

[32] B. Ummenhofer, H. Zhou, J. Uhrig, N. Mayer, E. Ilg, A. Dosovitskiy et T. Brox : Demon : Depth and motion network for learning monocular stereo. *In IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2017.

[33] Zhou Wang, Alan Bovik, Hamid Sheikh et Eero Simoncelli : Image quality assessment : From error visibility to structural similarity. *Image Processing, IEEE Transactions on*, 13:600 – 612, 05 2004.

[34] wikipedia : *rail gauge*. https://en.wikipedia.org/wiki/Track_gauge.

[35] Jianxiong Xiao, Owens Andrew et Torralba Antonio : Sun3d : A database of big spaces reconstructed using sfm and object labels. *In 2013 IEEE International Conference on Computer Vision*, 2013.

[36] Oliver Zendel, Katrin Honauer, Markus Murschitz, Daniel Steininger et Gustavo Fernandez Dominguez : Wilddash - creating hazard-aware benchmarks. *In Proceedings of the European Conference on Computer Vision (ECCV)*, September 2018.

[37] Oliver Zendel, Markus Murschitz, Marcel Zeilinger, Daniel Steininger, Sara Abbasi et Csaba Beleznai : Railsem19 : A dataset for semantic rail scene understanding. *In Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR) Workshops*, June 2019.

[38] Huangying Zhan, Ravi Garg, Chamara Saroj Weerasekera, Kejie Li, Harsh Agarwal et Ian Reid : Unsupervised learning of monocular depth estimation and visual odometry with deep feature reconstruction. *In The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2018.

[39] Hang Zhao, Orazio Gallo, I. Frosio et J. Kautz : Is l2 a good loss function for neural networks for image processing. *arXiv : Computer Vision and Pattern Recognition*, 2015.

[40] Chuanxia Zheng, Tat-Jen Cham et Jianfei Cai : T2net : Synthetic-to-realistic translation for solving single-image depth estimation tasks. *In Proceedings of the European Conference on Computer Vision (ECCV)*, pages 767–783, 2018.

[41] Bolei Zhou, Hang Zhao, Xavier Puig, Sanja Fidler, Adela Barriuso et Antonio Torralba : Scene parsing through ade20k dataset. *In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2017.

[42] Bolei Zhou, Hang Zhao, Xavier Puig, Tete Xiao, Sanja Fidler, Adela Barriuso et Antonio Torralba : Semantic understanding of scenes through the ade20k dataset. *International Journal on Computer Vision*, 2018.