Master of Science Thesis in Computer Vision
Department of Electrical Engineering, Linköping University, 2020

Unsupervised Learning for Structure from Motion

Erik Örjehag

Master of Science Thesis in Computer Vision

Unsupervised Learning for Structure from Motion

Erik Örjehag

LiTH-ISY-EX--YY/NNNN--SE

Supervisor: **Gustav Häger**
ISY, Linköpings universitet

Examiner: **Per-Erik Forssén**
ISY, Linköpings universitet

Division of Computer Vision
Department of Electrical Engineering
Linköping University
SE-581 83 Linköping, Sweden

Abstract

Perception of depth, ego motion and robust keypoints is critical for SLAM and structure from motion applications. Neural networks have achieved great performance in perception tasks in recent years. But collecting labeled data for supervised training is labor intensive and costly. This thesis explores recent methods in unsupervised training of neural networks that can predict depth, ego motion, keypoints and do geometric consensus maximization. The benefit of unsupervised training is that the networks can learn from raw data collected from the camera sensor, instead of labeled data. The thesis focuses on training on images from a monocular camera, where no stereo or LIDAR data is available. The experiments compare different techniques for depth and ego motion prediction from previous research, and shows how the techniques can be combined successfully. A keypoint prediction network is evaluated and its performance is compared with the ORB detector provided by OpenCV. A geometric consensus network is also implemented and its performance is compared with the RANSAC algorithm in OpenCV. The consensus maximization network is trained on the output of the keypoint prediction network. For future work it is suggested that all networks could be combined and trained jointly to reach a better overall performance. The results show (1) which techniques in unsupervised depth prediction are most effective, (2) that the keypoint predicting network outperformed the ORB detector, and (3) that the consensus maximization network was able to classify outliers with comparable performance to the RANSAC algorithm of OpenCV.

Acknowledgments

I would like to thank my supervisor Gustav Häger and examiner Per-Erik Forssén for their assistance and creative freedom letting me explore the topics that are of high interest to me. I would also like to thank the company Dyno Robotics for lending me the computer resources necessary to perform the experiments in this thesis.

Linköping, October 2020
Erik Örjehag

Contents

Notation **ix**

1 Introduction **1**
1.1 Motivation . . . 2
1.2 Research Questions . . . 2
1.3 Delimitations . . . 2

2 Background **5**
2.1 Convolutional neural networks and learning . . . 5

3 Related work **7**
3.1 Unsupervised depth and ego motion prediction . . . 7
3.2 Unsupervised keypoint prediction . . . 9
3.3 Unsupervised geometric consensus maximization . . . 10
3.4 Combined depth and feature point detection . . . 10

4 Method **11**
4.1 Datasets . . . 11
4.1.1 Sequence datasets . . . 12
4.1.2 Homographic adaptation dataset . . . 13
4.2 Consensus maximization dataset . . . 13
4.3 Comparing techniques for depth and ego motion prediction . . . 14
4.4 Evaluation metrics . . . 15
4.4.1 Depth error and accuracy metrics . . . 15
4.4.2 Camera ego motion error metric . . . 16
4.4.3 Keypoint error and score metrics . . . 16
4.4.4 Consensus maximization performance metrics . . . 17

5 Implementation **19**
5.1 Depth and ego motion prediction . . . 19
5.1.1 Architectures for depth and ego motion CNNs . . . 19
5.1.2 Differentiable depth image warping . . . 20
5.1.3 Loss functions . . . 21

5.1.4 Depth map normalization 23
5.1.5 Depth map up-scaling 23
5.1.6 Handling occlusions 24
5.1.7 Handling model limitations 25
5.2 Unsupervised keypoint learning 27
5.2.1 Loss function for keypoint learning 29
5.3 Unsupervised consensus maximization 33
5.3.1 Improving training convergence 39

6 Results **41**
6.1 Depth and ego motion 41
6.2 Keypoint detection 47
6.3 Consensus maximization 47

7 Discussion **51**
7.1 Results 51
7.1.1 Depth and ego motion 51
7.1.2 Keypoint detection 52
7.1.3 Consensus maximization 52
7.2 Method 52
7.2.1 Datasets 52
7.2.2 Evaluation metrics 53
7.3 Future work 53
7.4 Conclusions 55

Bibliography **57**

Notation

Math

Notation	Meaning
$\mathbb{R}$	The set of real numbers
m, M	Scalars are written in italics
$\mathbf{m}$	Vectors are written in lower case bold
$\mathbf{M}$	Matrices are written in upper case bold
$\mathbf{m}_i$	Row or column vector i (depending on context) of matrix $\mathbf{M}$
m_i	Element i of vector $\mathbf{m}$
m_{ij}	Element at row i and column j of matrix $\mathbf{M}$
$[m_{ij}]_{M\times N}$	Size of matrix is indicated with a subscript
$\mathbf{M}_{\text{name}}$	Subscripts can be used to give a unique name
$\mathbf{M}^{\text{name}}$	Superscript can also be used for the same purpose
$\delta_x\mathbf{I}$	Discrete derivative with respect to x-axis of matrix $\mathbf{I}$
$\mathbf{M}^T, \mathbf{m}^T$	Matrix or vector transpose
$\lvert m\rvert$	Absolute value of m
$\lVert\mathbf{m}\rVert$	Length of vector $\mathbf{m}$
$\mathbf{m}\cdot\mathbf{m}$	Vector dot product

Abbreviations

Abbreviation	Meaning
SFM	Structure From Motion
CNN	Convolutional Neural Network
SSIM	Structural Similarity index [26]
RGB	Red Green Blue, color space
RANSAC	Random Sample Consensus [4]
SVD	Singular Value Decomposition

1

Introduction

The aim of this thesis is to investigate the performance and implementation of some recently published techniques that use unsupervised training of neural networks in the structure from motion pipeline.

Imagine closing one eye and looking out into the world. It is trivial as a grownup human to detect motion and estimate how the head moves in relation to what is seen. Calculating camera movement from an image sequence is a well studied problem and is usually done by finding corresponding features in the images and calculating (using projective geometry) which camera movement can give rise to such correspondences and their relative movement between frames in the sequence.

Recent research has shown that it is possible to predict depth and relative motion from a sequence of images taken with a monocular RGB camera, up to an unknown scale factor. The training data is a sequence of unlabeled images with a small relative motion, for example looking out from the front window of a moving car. Given a target view and a few nearby views it is possible to train depth and pose predicting CNNs jointly using a combined loss function. The depth and pose predictions are used to warp nearby views to the target view and the loss is based on the visual similarity achieved after warping.

In addition to predicting depth and motion in an image sequence, it is also useful to extract feature points in the image that can be tracked over time in order to build a map of the world. In the process of extracting feature points, the system should be able to filter out points that are matched incorrectly with each other, so called outliers.

If depth prediction, camera motion, feature point detection and geometric consensus maximization are implemented in an unsupervised training framework it opens up the possibility to train all subsystems jointly where each part benefits from the rest. Collecting training data would be cheap, since no manual

annotation of the data is needed. But this is left as an area of future work and not covered in this thesis.

Self driving vehicles is a key area where neural network perception has a big impact. The adoption of autonomous vehicles could lead to more ride sharing alternatives, reducing the demand for owning a car, which in turn could reduce road congestion and oil consumption. [16]

1.1 Motivation

Here are three main points motivating future research into the use of unsupervised learning methods for visual structure from motion.

- Localization of autonomous vehicles is commonly achieved using lidar sensors due to their high accuracy and robustness, but using cameras instead comes with many other benefits[24]. For example, lower hardware cost and power consumption. Cameras are passive sensors which means that they do not interfere with each other. Lidars rely on spinning parts which can break if subjected to shaking or impact. Focusing on monoscopic vision instead of stereo vision again comes down to cost benefits.
- Obtaining labeled data for supervised training can be a tedious task, in that respect unsupervised methods are much more desirable. Labeling data is time consuming, expensive and prone to human errors. Collecting and storing data without labeling it is however comparably easy and inexpensive.
- This thesis focuses on depth, ego motion, keypoint prediction and consensus maximization. The motivation for this choice of unsupervised methods is the idea that in future work it might be possible to combine these methods to jointly learn all the tasks at the same time where each part of the system benefits from the others during training. The hope would be that the performance of the system would increase simply by collecting more unlabeled data.

1.2 Research Questions

1. How well do the unsupervised methods from previous research work on new datasets not tested in the original papers?
2. What are the performance gains of combining different methods from recent research in monocular depth and ego motion prediction?

1.3 Delimitations

The visual localization problem can be solved using, for example, a stereoscopic camera or a time of flight camera. But this thesis will only explore the use of a monoscopic, non depth sensing, RGB camera.

A full SFM pipeline can be conceptually divided into two parts, perception and map estimation. The perception block would process the sensor data and extract for example depth information, a pose update, and keypoint features that can be tracked over time. This is bundled as a keyframe that is passed on to the map estimation block that is responsible for building a map that is consistent over time with minimal drift[18].

This thesis will primarily focus on how unsupervised training can be applied to the perception block, including depth estimation and feature tracking, and will not be investigating anything to do with map estimation.

A few different unsupervised learning methods to predict depth and camera motion from a sequence of unlabeled images will be investigated. Additionally two specific unsupervised learning methods will be evaluated. Firstly how to predict feature points and their descriptors, and secondly how to perform geometric consensus maximization on the corresponding points.

2

Background

2.1 Convolutional neural networks and learning

The central method used in this project is a deep learning algorithm called convolutional neural networks (CNN for short). A CNN architecture can successfully capture the spatial dependencies in an image through convolutional filtering operations with kernels of learnable weights and biases. [10]

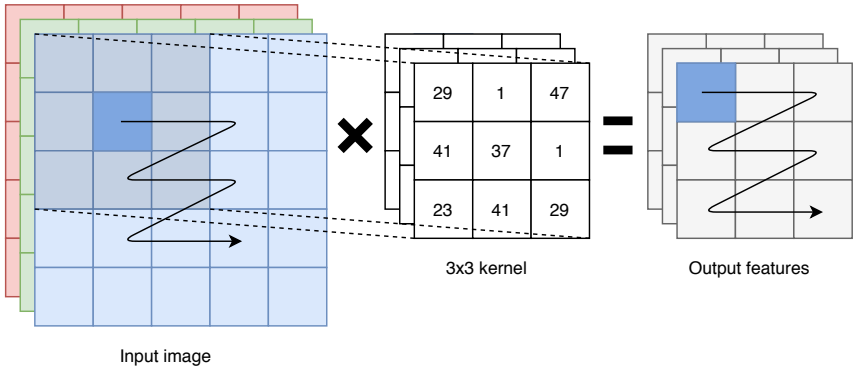

Figure 2.1: *Convolutional filtering operation with a 3 channel RGB image and 3x3 kernel.*

In Figure 2.1 a convolutional filtering operation over an image is illustrated. The matrix kernel is moved in a row by row pattern and is multiplied by a patch of the image to get a value for the output cell.

In order to form a deep neural network multiple filtering operations are chained sequentially with nonlinear activation functions between them. A deep neural

network usually consists of many such layers of filtering operations and activation functions [8]. This concept is illustrated in Figure 2.2.

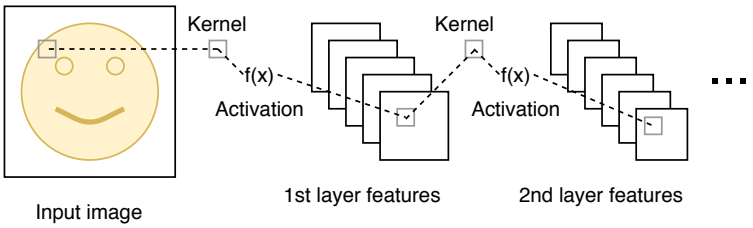

Figure 2.2: *Multiple layers chained with each other to form a deep network.*

The activation functions need to be differentiable because the derivatives are used in the learning process. Some common nonlinear activation functions are shown in Figure 2.3.

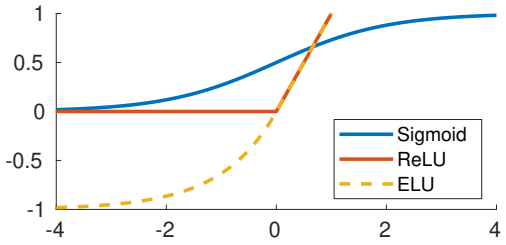

Figure 2.3: *A few common nonlinear activation functions.*

The features in the deep neural networks are used to formulate a loss function. The loss defines an objective that we want the network to learn. This means that the process of learning becomes a task of updating the filter kernels in such a way that the loss function is minimized. If the loss function is decreasing during the training process it means that the network is learning [9]. It is crucial that the loss function describes the problem accurately, otherwise the network will not learn the correct behavior. The weights in the network are updated using a method called back propagation. The algorithm computes the gradient of the loss function with respect to the weights in the network for a single input example from the training data using the derivative chain rule, which can be done very efficiently. Updating the weights to minimize the loss function can then be done using gradient decent [11]. As the network is fed with more input examples from the training data, the network slowly learns the correct weights that minimizes the loss function as desired.

3

Related work

In this chapter some of the important contributions of previous research papers are summarized. Firstly a list of papers in the field of unsupervised depth and ego motion prediction is presented. Secondly a paper on unsupervised feature point prediction. Thirdly a paper on unsupervised geometric consensus maximization that was implemented in this thesis. Finally a paper on combining depth and feature point prediction, albeit using classical consensus maximization that is not learned by a neural network.

3.1 Unsupervised depth and ego motion prediction

All the papers in this section are discussed in the order of publication to show a timeline of progress in the field of depth and ego motion prediction.

In this paper[6] the authors present MonoDepth, with an implementation available in Tensorflow on Github. In this work the depth is predicted using an encoder-decoder type network, but the relative motion between frames is not estimated at all. The KITTI dataset provides stereo image pairs which are used during training, and the relative transformation between the left and right cameras is known. Using only the left image as input to the network both disparity maps for the left and right images are predicted. The two disparity maps are used to project the left image into the view of the right image and vice versa. This can be seen as a precursor to the papers discussed later which uses only a monocular camera and several frames over time to train a depth predicting network and pose predicting network jointly. The L1 norm of the per pixel photometric error as well as SSIM [26] are computed and added to the loss. An additional loss term encourages the left disparity to be equal to the right disparity projected into the left camera viewpoint. Because the photometric error does not work well on low textured regions an edge aware smoothess term is added to propagate the depth

values from nearby areas in the disparity map. The method produces metrically accurate results because the baseline and focal length of the cameras are known.

In this paper[25] the authors present SfMLearner with an official implementation in Tensorflow on Github. Contrary to the previous paper[6] only a monocular sequence of images from the KITTI dataset is used during training. In the stereo case the relative pose between the left and right cameras was known, but with this monocular dataset the pose between subsequent frames is unknown. The authors train a pose predicting and depth predicting network simultaneously with a joint loss function to solve this problem.

During training 3 subsequent frames are considered at a time. The frame $\mathbf{I}_{t-1}$ and frame $\mathbf{I}_{t+1}$ are called the source frames and the frame $\mathbf{I}_t$ is called the target frame. The target frame is input to the depth network which estimates a disparity map. The two source frames are fed through a pose estimating network one after each other together with the target frame to find the relative transformations $\mathbf{T}_{t\to t-1}$ and $\mathbf{T}_{t\to t+1}$.

The authors add an explainability mask to the photometric error term to account for violations of the static scene assumption. The view synthesis formulation implicitly assumes that the scene does not contain moving objects, that there are no occlusions between the target and source frames, and that the surfaces are Lambertian so that the photo-consistency error of RGB values is meaningful. In order to predict the explainability mask an additional CNN is used. The network has no explicit supervisory signal but is encouraged to be non-zero with a regularization term using a cross-entropy loss with a constant label 1 at each pixel location. This makes the network minimize the view synthesis objective but is allowed some slack due to factors not considered by the model. In later work it was shown that the explainability mask does not help to improve results that much and is often ignored.

To tackle the problem with textured areas and non-Lambertian surfaces, a smoothess term is used. An edge aware smoothess term was not used like in previous work[6] but a penalty on the second order gradient of the depth map was used instead. This unfortunately makes edges very fuzzy in the results compared to using the edge aware smoothness loss.

In this paper[20] the authors train a depth estimating network on only stereo image pairs. After the depth predicting network has been trained the pose network from [25] is trained, using results from the already trained depth predicting network. This means that the depth and pose networks are not trained with a joint loss function but are instead trained separately. The main contributions from this paper is a new subpixel-convolution operation that super-resolve disparities from their low-resolution outputs, thereby replacing the up-sampling layers typically used in the disparity decoder network. The method additionally uses differentiable flip-augmentation to remove edge artifacts on the left and right edges of the depth map seen in previous work using stereo image pairs during training. To handle occluded pixels between the left and right images an occulsion regularization loss term is added to encourage background depths (low disparaty).

In this paper[12] the authors present PacknetSfM. The main contribution is

a new network architecture with packing and unpacking blocks replacing down and up sampling. The new packing blocks use space to depth transformations and 3d convolutions. The authors claim that the new packing and unpacking blocks are better at perserving resolution than standard down and upsampling. As the method is not adopted in later work, it is unclear if the new packing and unpacking approach is effective. The second contribution is a loss on the camera velocity that makes it possible for the monocular depth estimation to be metrically accurate. The velocity of the car is assumed to be known in the training dataset. The third contribution is a mask on pixels that do not change between frames. These pixels are assumed to be part of objects that are stationary with respect to the camera. Such pixels can occur for example if the car dashboard is visible in a frame, or other vehicles are moving at a similar speed nearby.

In a follow up paper[7] the authors present MonoDepth2. They propose mainly two contributions. Firstly, instead of taking the average of the reprojection errors from all source frames given a target frame they use the minimum. This makes it so that if a feature is occluded in one source image but not in the other, the errors will not be averaged together but instead the error from the source frame that is not occluded will be used. Secondly, instead of calculating the loss for each depth scale in the decoder, all the depth maps are up-sampled to the original target image size when computing the loss. This way a single pixel in the low resolution layer of the decoder will predict the depth of a patch of pixels in the originally sized input image.

3.2 Unsupervised keypoint prediction

Collecting ground truth data to train a neural network for keypoint detection is cumbersome. What constitutes a keypoint is hard to define clearly and consistently for a human annotator.

SuperPoint[3] uses a unsupervised approach to learn prediction of points and their descriptors. The network is first trained on a synthetic dataset of simple geometric shapes. The pseudo ground truth points are defined as the corners and junctions in the synthetic images. This pre-trained network is then trained on "real" images in a siemese network setup using a method they call homographic adaptation. The two siamese siblings are fed images transformed by random but known homographies and the output from the networks are compared in the loss function. The output of SuperPoint is a heatmap where each pixel describes the "point-ness" of that pixel, and a descriptor map. To keep the model fast both the point and descriptor maps are predicted semi-dense grid, one cell for each 8 pixel patch. The maps are up-sampled to the original image size using bicubic interpolation, and the descriptors are normalized.

UnsuperPoint[2] is heavily inspired by SuperPoint but does not need to be pre-trained on a synthetic dataset, instead it is trained in one round of training on real images. It uses a similar method of using siamese networks and homography adaptation during traning, but the output from the network is different. The network uses regression of actual point position coordinates instead of a heat

map, incorporates non-maximum suppression, predicts scores for each keypoint and a sparse descriptor map that is sampled and interpolated using the point positions.

3.3 Unsupervised geometric consensus maximization

Consensus maximization is an important strategy in 3D vision problems for robust geometric model estimation from measurements including outliers. The classical method of *Random Sample Consensus*, or RANSAC[4] for short, is widely popular with great success. But replicating the same generic behavior using supervised training of neural networks has proven difficult. Unsupervised methods have a huge potential to generalize to any unseen data distribution and are in this context very desirable. Another paper[21] introduces just such an unsupervised method of consensus maximization for 3D vision problems. Using the relationship between the set of inliers, and the subspace of polynomials representing the space of target transformations a model fitting cost can be can be defined without knowing the specific parameters of the geometric transformation. During learning, the loss is defined such as to learn the largest set of inliers with a low model fitting cost. The geometric model parameters can then be extracted from the polynomials after network has learnt to distinguish inliers and outliers.

3.4 Combined depth and feature point detection

In a paper[23] the authors combine the previously discussed MonoDepth2[7] for depth prediction, but also adds keypoint learning from SuperPoint[3] into the pipeline. The researchers train the depth, keypoint and pose estimating networks jointly, making them benifit from each other and achieve state of the art performance. However, the step that finds possible outlier keypoints, is not differentiable and is not trained jointy with the depth and keypoint networks. The step that calculates the model parameters of the geometric relationship between corresponding keypoints also requires an initial guess that is not differentiable.

4

Method

This chapter describes the methods used to answer the research questions. The first question relates to the datasets used during training and testing of the networks. The second question relates to the performance benefits of applying specific techniques from previous research in unsupervised monocular depth and ego motion prediction.

To begin with the chapter summarizes which datasets where used in this thesis and how the choice relates to the first research question. Following that an explanation of which specific techniques for monocular depth prediction are evaluated in the results chapter, and the method used to combine them. Lastly a detailed description of all the error and accuracy metrics that are presented in the results chapter. A good understanding of what the metrics imply is needed to answer both research questions.

4.1 Datasets

The neural networks are trained and evaluated on images from 2 different datasets, KITTI[5] and Lyft[15]. Both datasets are preprocessed to remove frames where the camera is not moving. This is important because if there is no movement between frames then no depth information can be inferred during training when using monoscopic data. In the Kitti dataset, the images from the left and right cameras are treated as separate image sequences to yield more training data. The images are resized to height $H = 128$ and width $W = 416$ pixels, and the intrinsic camera matrix $\mathbf{K}$ is updated accordingly.

4.1.1 Sequence datasets

To train the networks used for depth and ego motion prediction the images from Kitti and Lyft at loaded in triplets of subsequent frames which are used during training of the network. We denote an image by $\mathbf{I}$, which is a $3 \times H \times W$ matrix of RGB pixel colors. To test the network, the lidar data and ground truth ego motion from Kitti and Lyft are used. The lidar data is converted to a sparse depth map, that can be compared to the depth map predicted by the network. We denote a depth map by $\mathbf{D}$, which is a $1 \times H \times W$ matrix of depth values for each pixel. The ground truth lidar and ego motion data is only used during testing, not during training. A sample of 3 subsequent frames can be seen in Figure 4.1. We denote ego motion by $\mathbf{T}$, which is a 3×4 rigid 3D transformation matrix.

Figure 4.1: *The dataloader loads 3 subsequent frames in a sequence. The figure shows from top to bottom 3 frames from Kitti, $\mathbf{I}_{t-1}$, $\mathbf{I}_t$, $\mathbf{I}_{t+1}$ with the sparse depth map overlaid on frame $\mathbf{I}_t$.*

Kitti contains 124 sequences, and Lyft contains 148 sequences. For each dataset the sequences are split, approximately 90% is used for training and 10% for testing. A detailed breakdown of the data split can be seen in Table 4.1.

	Sequences / Samples	
	Train	Test
Kitti	110 / 16542	12 / 11349
Lyft	134 / 3759	14 / 1735

Table 4.1: *The dataset sequences are split into a training set and a testing set. Approximatly 10% of the sequences are used for testing.*

4.1.2 Homographic adaptation dataset

To train the network that predicts keypoints, images are read one by one from the Kitti or Lyft datasets. The image is fed trough two branches, in branch A the image is not modified, and in branch B the image is transformed by a random homography (Figure 5.10). The authors of UnsuperPoint refer to this technique as homographic adaptation.

How to generate the random homography used during training is not described in the UnsuperPoint paper, but is an important part of the method. The method used in this thesis generates random homographies from 5 parameters α_{rotation}, $\alpha_{\text{translation}}$, α_{scale}, α_{sheer} and $\alpha_{\text{perspective}}$. The parameters controls the maximum transformation for each aspect of an homography. The final homographgy is constructed from parts as follows.

$$\mathbf{H} = \mathbf{H}_{\text{affine}}\mathbf{H}_{\text{sheer}}\mathbf{H}_{\text{perspective}} \tag{4.1}$$

Assume $u_n \sim \text{U}(-1, 1)$ are random uniform variables in the range -1 to 1.

$$\mathbf{H}_{\text{affine}} = \begin{pmatrix} \cos(r) * s & -\sin(r) & t_{\text{x}} \\ \sin(r) & \cos(r) * s & t_{\text{y}} \\ 0 & 0 & 1 \end{pmatrix}, \text{with} \begin{cases} r = u_1 * \alpha_{\text{rotation}} \\ s = u_2 * \alpha_{\text{scale}} + 1 \\ t_{\text{x}} = u_3 * \alpha_{\text{translation}} \\ t_{\text{y}} = u_4 * \alpha_{\text{translation}} \end{cases} \tag{4.2}$$

$$\mathbf{H}_{\text{sheer}} = \begin{pmatrix} 1 & s & 0 \\ s & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}, \text{with } s = u_5 * \alpha_{\text{sheer}} \tag{4.3}$$

$$\mathbf{H}_{\text{perspective}} = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ p & p & 1 \end{pmatrix}, \text{with } p = u_6 * \alpha_{\text{perspective}} \tag{4.4}$$

The output from the network is used together with the random but known homography to formulate the loss function.

4.2 Consensus maximization dataset

The consensus maximization network is trained on the output from the keypoint network. During training the weights of the keypoint net are frozen. Figure 4.2 illustrates the flow of data in the training and evaluation pipeline. Notice that the ground truth homography is not used in the loss function, and is only used in the evaluation metric to test the performance of the predictions.

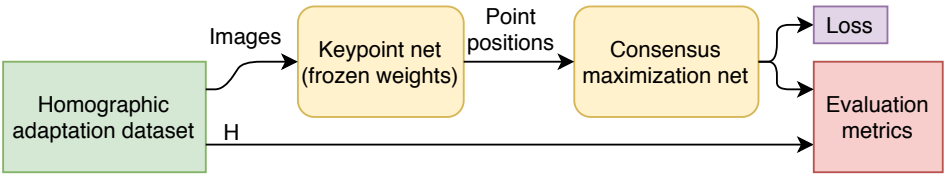

Figure 4.2: *Illustrates the flow of data in the training and evaluation pipeline of the consensus maximization network.*

4.3 Comparing techniques for depth and ego motion prediction

To answer the second research question, a system of enabling and disabling terms in the loss function was implemented. The effect on performance for a few particular loss terms discussed in the related work was measured. By enabling a selection of loss terms (Table 6.1) in different experiments (Table 6.2) their respective contribution to the results can be observed. The command line options to the program used to train the models are listed in Table 4.2. An deeper explanation of these techniques can be found in the chapter 5.

Command line option	Name	Description
`--net`	Net	Network architecture (SfMLearner or Monodepth2).
`--dataset`	DS	Dataset for training (Kitti or Lyft).
`--explain-mask`	Expl	Filter pixels using explainability mask (section 5.1.7)
`--stationary-mask`	Stat	Filter pixels using stationary pixels mask (section 5.1.7)
`--ssim`	SSIM	Use SSIM in combination with L1 in the photometric error term (section 5.1.3).
`--depth-map-norm`	Norm	Normalize the depth map (section 5.1.4)
`--edge-aware`	Edge	Use edge aware depth smoothness loss term (section 5.1.3)
`--upscale`	US	Use up-scaling of the smaller depth maps in the decoder (section 5.1.5).
`--min-loss`	Comb	Combine loss from $t-1$ and $t+1$ with `min` instead of `avg` (section 5.1.6).

Table 4.2: *A list of command line option for the PyTorch training script. The column Name contains the naming used for each technique when they are used to form the configurations in Table 6.1 in chapter 6 Results.*

4.4 Evaluation metrics

This section describes the metrics used to evaluate depth, ego motion, feature point and consensus maximization predictions. These specific metrics where chosen because they are also used in the related works in this field, which makes the results comparable to other papers.

4.4.1 Depth error and accuracy metrics

The depth error and accuracy metrics are sparse and only calculated for the pixels of which there exists a ground truth laser measurement in the dataset. The values are averaged over all laser measurements and frames in the test split of the dataset. An error of 0 and an accuracy of 1 is optimal, but can never be achieved in practice.

The depth is predicted relative to an unknown scale, and the ground truth is measured in meters. To alleviate this issue the depth predictions are scaled so that their median is the same as the median of the ground truth for each frame. This does not guarantee that the unit of the predictions is meters, but at least it makes the scales somewhat similar.

The error metrics used in the results chapter are referred to as *Abs Rel*, *Sq Rel*, *RMSE* and *RMSLE*. Following is an explanation of all the metrics used in the results chapter. We denote the ground truth y_n and the prediction $\hat{y}_n$.

The *Abs Rel* error is based on *MAE* which is the mean of the absolute errors which means that it has the same unit as the errors, and is conceptually quite easy to interpret.

$$\text{MAE} = \frac{\sum_{n=1}^{N} |y_n - \hat{y}_n|}{N} \tag{4.5}$$

Because the depth from the neural network is without unit and only predicted relative to an unknown scale, a variation on the *MSE* metric called *Abs Rel* is used.

$$\text{Abs Rel} = \frac{\sum_{n=1}^{N} \frac{|y_n - \hat{y}_n|}{y_n}}{N} \tag{4.6}$$

The *MSE* metric is the mean of squared errors, for an unbiased estimator it represents the variance of the errors.

$$\text{MSE} = \frac{\sum_{n=1}^{N} (y_n - \hat{y}_n)^2}{N} \tag{4.7}$$

Once again, because the depth is predicted relative to an unknown scale a variation on *MSE* called *Sq Rel* is used instead.

$$\text{Sq Rel} = \frac{\sum_{n=1}^{N} \frac{(y_n - \hat{y}_n)^2}{y_n}}{N} \tag{4.8}$$

The *RMSE* metric is the square root of the mean of squared errors. For an unbiased estimator it represents the standard deviation of the errors. Because

the errors are squared in the *RMSE* metric it is more sensitive to outliers that for example *MSE*.

$$\text{RMSE} = \sqrt{\frac{\sum_{n=1}^{N} (y_n - \hat{y}_n)^2}{N}} \tag{4.9}$$

The *RMSLE* metric is similar to *RMSE* but is useful because it penalizes large errors less when both the actual and predicted values are large.

$$\text{RMSLE} = \sqrt{\frac{\sum_{n=1}^{N} (\log y_n - \log \hat{y}_n)^2}{N}} \tag{4.10}$$

To measure the depth accuracy, in the range from 0 to 1, the following metric is used.

$$a_\gamma = \frac{\sum_{n=1}^{N} (\max(\frac{y_n}{\hat{y}_n}, \frac{\hat{y}_n}{y_n}) < \gamma)}{N}, \text{ for } \gamma \in \{1.25, 1.25^2, 1.25^3\} \tag{4.11}$$

This should be interpreted as the ratio of predictions that is within the ratio of γ relative to the ground truth.

4.4.2 Camera ego motion error metric

The error of the camera motion predictions are measured using *RMSE*. But instead of taking the mean error over all poses in a sequence, the alignment error is calculated part wise over a track length of only 5 poses. The final error is presented as the mean *RMSE* of all parts. Each track part from the prediction is transformed to have its first pose coincide with the ground truth in a common origo. This is done so that the first pose in the track part of both the prediction and ground truth is the 4×4 identity matrix. Because the ego motion is predicted relative to an unknown scale and the ground truth is in meters, we scale each predicted track part to have similar scale to the ground truth track part.

4.4.3 Keypoint error and score metrics

The unsupervised keypoint network was compared with the ORB feature detector from OpenCV. To messure the performance differences 3 different metrics where used, repeatability score (RS), localization error (LE), matching score (MS), and matching ratio (MR).

Before any of the metrics are calculated the points are first filtered to remove all points within 10px of the edges, see Figure 4.3. This is done to not include detections where the black background meets the border of the transformed image.

Figure 4.3: *The black and yellow stripes illustrates the are where keypoint detections are filtered out and discarded.*

To calculate the set of ground truth correspondences, the points in the left image is transformed by the known homography **H**. A point in the left image is corresponding with a point in the right image if they are their closes neighbor and the distance is less than 3px after the transformation. This set of ground truth correspondences is used in all metrics. If the images contain a different amount of points, the largest set of possible correspondences that could have been found is $N_{\text{tot}} = \min(N_{\text{left}}, N_{\text{right}})$. The number of ground truth correspondences (closer than 3px) is $N_{\text{gt}} \leq N_{\text{tot}}$.

Repeatability score (RS) is a measure on how good a method is at repeatedly highlighting the same physical features in the scene but in different images. Because the keypoint algorithm is only fed one image at the time, it is important that there is consistency in what physical features are selected in the image, otherwise their will be very few actual correspondences.

$$\text{RS} = \frac{N_{\text{gt}}}{N_{\text{tot}}} \tag{4.12}$$

Localization error (LE) is a measure of the average distance error between the ground truth corresponding points. It will always be less than 3px because that is our definition of a correspondence, but the ideal is 0px.

Matching score (MS) is a measure on how many keypoints are matched correctly based on their descriptors. The set of points that are both ground truth correspondences and are corresponding based on their predicted descriptors are said to be correctly matched. The number of correctly matched points are N_{correct}.

$$\text{MS} = \frac{N_{\text{correct}}}{N_{\text{tot}}} \tag{4.13}$$

Matching ratio (MR) is very similar to matching score. To get a high MS requires both good repeatability and good descriptors, but getting an MR requires only good descriptors.

$$\text{MR} = \frac{N_{\text{correct}}}{N_{\text{gt}}} \tag{4.14}$$

4.4.4 Consensus maximization performance metrics

The unsupervised consensus maximization network that predicts homographies was compared with the *findHomography()* function in OpenCV.

The homography error (HE) is calculated as the average distance between the corners of the image when it is transformed by the estimated homography compared to the ground truth homography, see Figure 4.4.

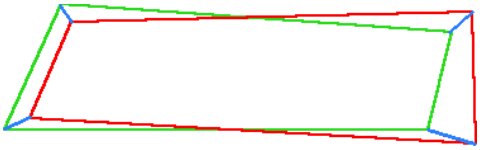

Figure 4.4: *The corners of the image transformed by the ground truth homography and the predicted homography is shown in green and red respectively. The homography error (HE) is calculated as the average of the distances shown in blue.*

In order to evaluate the methods ability to distinguish inliers and outliers among the points a confusion matrix is used, see Table 4.3. The ground truth set of inliers, or ideal, is the same as the one calculated in section 4.4.3. The confusion matrix shows the ratio of true positives, true negatives, false positives and false negatives in the inlier classifications. The confusion matrix is normalized on the axis of ground truth so that the sum of true positives and false negatives adds up to 1, and the sum of the false positives and true negatives also adds up to one.

		Predicted		
		Inlier	Outlier	Total
Actual	Inlier	TP	FN	TP + FN = 1
	Outlier	FP	TN	FP + TN = 1
	Total	TP + FP	FN + TN	2

Table 4.3: *Structure of confusion matrix for inlier classification.*

5
Implementation

This chapters describes the implementation done in PyTorch for this thesis. The first part details the methods evaluated in depth and ego motion prediction, the second part describes the implementation of unsupervised keypoint prediction, and the last part describes how the consensus maximization network was implemented.

5.1 Depth and ego motion prediction

This section explains the implementation that was done in this thesis project for the depth and ego motion predicting networks.

5.1.1 Architectures for depth and ego motion CNNs

In order to predict depth and motion from monocular images two different CNN architectures are examined. Both are encoder-decoder type architectures, and their general layout is illustrated in Figure 5.1.

The first architecture is referred to as SfMlearner[25]. It uses a DispNet[19] architecture to predict depth maps at four different scales.

The second architecture is referred to as Monodepth2[7]. It uses a ResNet18 architecture instead of a DispNet architecture to predict depth estimates. This architecture is smaller resulting in faster training and evaluation.

Both SfMlearner and Monodepth2 uses a separate network to predict poses between frames. The pose network has a ResNet18[13] architecture, but the decoder is modified to predict pose updates in an Euler-angle-axis representation.

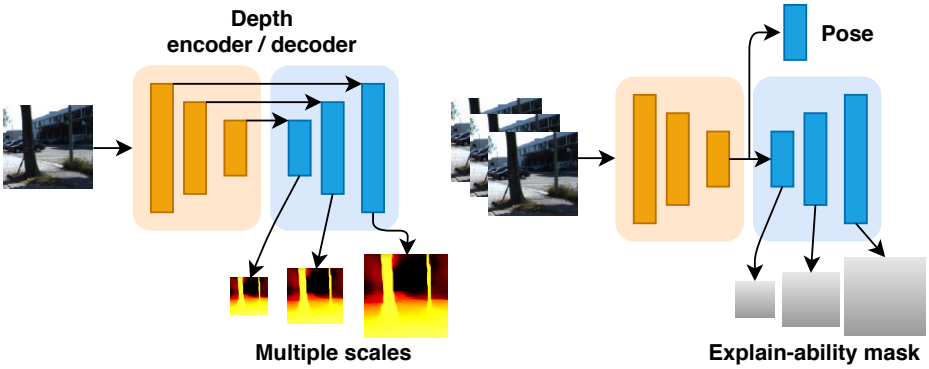

Figure 5.1: *High level diagram of the network architectures used for depth and ego motion prediction. The layers from the depth encoder are concatenated into the layers of the decoder. Depth maps are computed at multiple scales in the decoder and are all used in the loss function. A separate network that takes as input 3 subsequent frames predicts the poses $\boldsymbol{T}_{t\to t-1}$ and $\boldsymbol{T}_{t\to t+1}$ between the target and nearby reference frames. The pose network shares encoder with the explain-ability mask predicting network (section 5.1.7).*

5.1.2 Differentiable depth image warping

The core component of unsupervised depth learning is the differentiable depth image warp operation in the loss function of the CNN networks. The goal is to reconstruct the target image from pixel values in the source image using the predicted depth and ego motion. If the predictions are good, then the reconstruction will also be good. If the loss is based on the quality of the reconstruction then the loss will be a good measure on how well the network has learned to predict depth and ego motion. To perform the reconstruction we need a few components. Firstly the intrinsic camera matrix:

$$\mathbf{K} = \begin{pmatrix} f_x & s & x_0 \\ 0 & f_y & y_0 \\ 0 & 0 & 1 \end{pmatrix} \tag{5.1}$$

where f is the focal length, s is the skew coefficient (often zero), and x_0 and y_0 represent the principal point. Secondly we need the the predicted depth $\mathbf{D}_t(\mathbf{p}_t)$ of pixel $\mathbf{p}_t$ of the target (current) frame. Thirdly we need the predicted ego motion in the form of a transform matrix $\mathbf{T}_{t\to s}$ from the target to the source (next or previous) frame in the sequence:

$$\mathbf{T}_{t\to s} = \begin{pmatrix} \mathbf{R} & \mathbf{t} \\ 0 & 1 \end{pmatrix} \tag{5.2}$$

Given these, we can project the position of the target pixel $\mathbf{p}_t$ onto the source image $\mathbf{I}_s$. We call this new position $\mathbf{p}_s$, and it is calculated in homogeneous coordinates as:

$$\mathbf{p}_s \sim \begin{pmatrix}\mathbf{K} & \mathbf{0}\end{pmatrix} \mathbf{T}_{t\to s} \mathbf{D}_t(\mathbf{p}_t) \mathbf{K}^{-1} \mathbf{p}_t \tag{5.3}$$

The pixel position $\mathbf{p}_s$ is however continuous and in order to sample the discrete source image $\mathbf{I}_s$ a differentiable bilinear sampling method is used[14]. The value at the continuous sampling point $\mathbf{p}_s$ is interpolated from its four discrete neighbors. This process is illustrated in Figure 5.2.

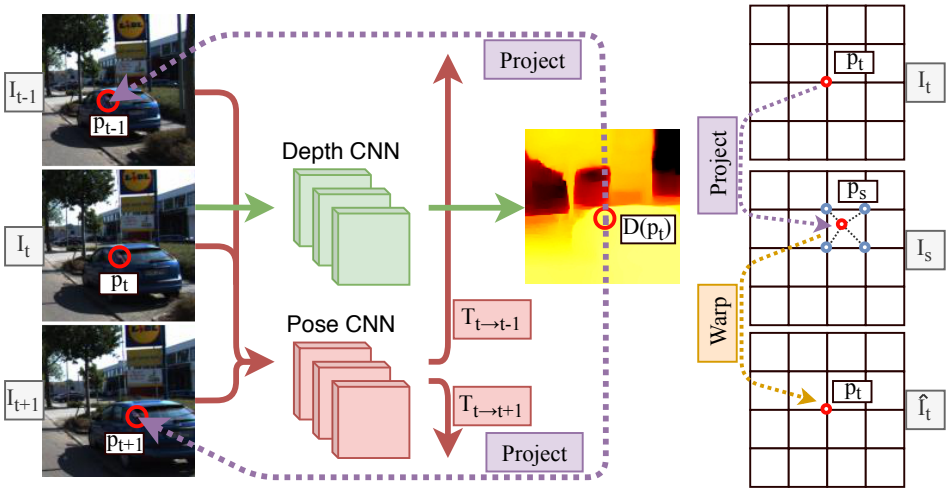

Figure 5.2: The CNN predicts the depth map $\mathbf{D}$ of the target image $\mathbf{I}_t$, and also the relative movement, $\mathbf{T}_{t\to t-1}$, $\mathbf{T}_{t\to t+1}$ between the target image and the source images. Each pixel p_t in the target image is projected onto a position in the source images which are sampled using bilinear interpolation. This should recreate the appearance of the target image but with pixels sampled from the source image. An appearance similarity metric between the original target image and the recreated target images can be used as the loss function for the CNN to learn to accurately predict correct depth and movement.

5.1.3 Loss functions

In order to learn the objective of depth and ego motion prediction, different combinations of the following loss terms where evaluated.

To measure the similarity of the target image $\mathbf{I}_t$ and the reconstructed image $\hat{\mathbf{I}}_t$ a photometric loss term defined as the mean of the absolute value of the difference of pixel intensities of the two images was used.

$$\mathcal{L}_p(\mathbf{I}_t, \hat{\mathbf{I}}_t) = \text{mean}(|\mathbf{I}_t - \hat{\mathbf{I}}_t|) \tag{5.4}$$

Another photometric loss term evaluated in this thesis is based on structured similarity, referred to as SSIM[26]. It was originally developed to measure the quality of digital television, comparing a compressed digital image to the original distortion-free image.

$$\mathcal{L}_{ssim}(\mathbf{I}_t, \hat{\mathbf{I}}_t) = \text{mean}(\text{clamp}(\mathbf{0}, \mathbf{1}, \frac{\mathbf{1} - \text{SSIM}(\mathbf{I}_t, \hat{\mathbf{I}}_t)}{2})) \tag{5.5}$$

$$\text{SSIM}(\mathbf{X}, \mathbf{Y}) = \frac{(2\mu_\mathbf{X}\mu_\mathbf{Y} + C_1)(2\sigma_\mathbf{XY} + C_2)}{(\mu_\mathbf{X}^2 + \mu_\mathbf{Y}^2 + C_1)(\sigma_\mathbf{X}^2 + \sigma_\mathbf{Y}^2 + C_2)} \tag{5.6}$$

with $C_1 = 1e-4$ and $C_2 = 9e-4$. To compute the per-patch mean $\mu_\mathbf{X}$ and standard deviation $\sigma_\mathbf{X}$, a 1 pixel reflection padding was first used on the edges of the input images and then a 3×3 average pool filter with stride 1 was used to get the mean. Then $\sigma_\mathbf{X} = \mu_{\mathbf{X}^2} - \mu_\mathbf{X}^2$.

The two above mentioned photometric loss terms can also be combined and balanced using $\mathcal{L}_{ps}(\mathbf{I}_t, \hat{\mathbf{I}}_s) = \alpha\mathcal{L}_{ssim} + (1-\alpha)\mathcal{L}_p$. In the experiments of this thesis $\alpha = 0.85$ was used, which is the same value used in the original Monodepth2 paper. As the network learns over time to predict more accurate depth and ego motion, the reconstruction loss will decrease. The loss per individual pixel is illustrated in Figure 5.3, which shows the loss from the same sample but after 1 and 30 epochs of training respectively.

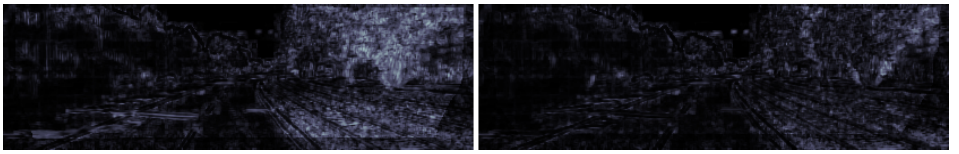

Figure 5.3: *The images illustrates the photometric reconstruction loss $\mathcal{L}_{ps}$ for each pixel in a reconstructed image for the same frame but after different length of training. The left image shows the loss after 1 epoch of training and the right image shows the loss after 30 epochs. The reconstruction loss should decrease during training as the network learns to predict better depth maps, which is what we see.*

To propagate the depth from textured regions to regions of uniform color a depth smoothness loss term is used. The first alternative is a loss on the second derivative of the depth intensities. This loss term will discourage the network to predict fluctuating depth values in regions of uniform color such as the pavement, that should be smooth. The drawback of this technique is that changes in depth values are penalized equally in regions where there are a lot of detail that therefore could be lost in the depth map.

$$\mathcal{L}_{smooth}(\mathbf{D}_t) = \text{mean}(|\delta_x^2\mathbf{D}_t| + |\delta_y^2\mathbf{D}_t|) \tag{5.7}$$

The second alternative investigated was an edge aware smoothness loss that weights the first order derivative of the depth map with the exponential of the first derivative of the pixel intensities. This method allows large changes in depth near sharp features in the image but penelizes changes in depth in smooth regions, see Figure 5.4. The method showed great promise, resulting in sharper

edges because the smoothness term is weighted to mostly affect areas with small intensity derivative.

$$\mathcal{L}_{edge}(\mathbf{D}_t) = \text{mean}(|\delta_x \mathbf{D}_t| e^{-|\delta_x \mathbf{I}_t|} + |\delta_y \mathbf{D}_t| e^{-|\delta_y \mathbf{I}_t|}) \tag{5.8}$$

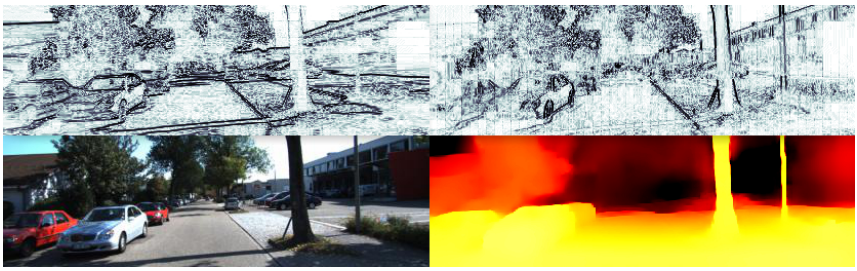

Figure 5.4: *The top row of images illustrates the edge weighting terms $e^{-|\delta_y \mathbf{I}_t|}$ and $e^{-|\delta_x \mathbf{I}_t|}$ respectively. The weight is near 0 at the edges of tree trunks and near 1 on the pavement. This will preserve the details in the depth map around trees but keep the pavement smooth.*

5.1.4 Depth map normalization

The predicted depth is scale invariant which can make it difficult to balance the loss terms with the correct weights. To alleviate this is issue the depth map can be normalized before it is used in the loss.

$$\hat{\mathbf{D}}_t = \frac{\mathbf{D}_t}{\text{median}(\mathbf{D}_t)} \tag{5.9}$$

5.1.5 Depth map up-scaling

The smaller depth maps in the depth decoder of the network are all used in the loss. Early work in this area down-scaled the target image to fit the size of the smaller depth maps when used in the loss function. But in MonoDepth2[7] it was proposed to instead up-scale the depth maps to the original target image size, see Figure 5.5.

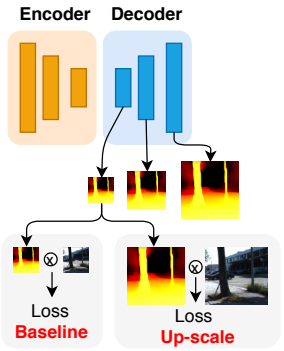

Figure 5.5: *The depth maps can optionally be up-scaled in the loss function.*

5.1.6 Handling occlusions

In the SfMLearner paper the photometric loss is calculated for the previous and next frames compared to the current in the sequence. The pixel-wise average across the frames is then used. This causes problems if a pixel is for example occluded in the previous frame, but visible in the current and next frame. In this situation the average loss will be large even though a correct depth and transformation has been predicted, because of the occluded pixel. Instead Monodepth2 suggests to pick the minimum per pixel error over the frames which alleviates this issue, see Figure 5.6. Both techniques where implemented in this thesis and compared in the experiments.

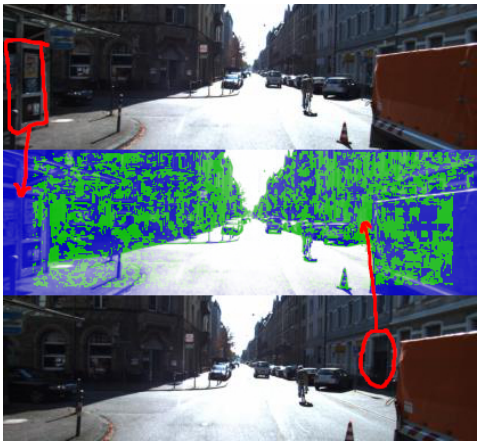

Figure 5.6: By picking the per pixel minimum reprojection error the issue created by occluded pixels can be alleviated. The top image is the previous frame, the middle image is the current target frame and the bottom image is the next frame in the sequence. If the minimum reprojection error can be found in the previous frame then it is colored blue, if its from the next frame it is green. Because the door to the right is occluded by the orange truck in the previous frame, the reprojection loss from the next frame is used instead where the door is visible. The wall to the left is outside the boundaries of the next image, so the reprojection error from the previous frame is used instead.

5.1.7 Handling model limitations

In order to optimize using the photometric reprojecton error as the loss function two assumptions must hold. Firstly the scene must be static, meaning all objects in the scene must be still except the moving camera. Dynamic objects will cause problems. Secondly there must be photometric consistency between frames for the photometric error to make sense. This means that non Lambertian surfaces, change in lighting, and change in exposure between frames will cause problems. Two different methods of dealing with this issue was implemented and evaluated in this thesis.

Explainability mask The authors of [25] tackle the model limitations by having a CNN predict what pixels are valid to use in the photometric loss function, see Figure 5.7. It shares the encoder of the pose predicting network but branches of into a different decoders which estimates a mask of the valid/explainable pixels. The loss function for the mask is the cross entropy loss compared to a mask filled with ones. The photometric loss function is augmented to include the explainability mask removing pixels that cannot be explained by the predicted depth and transformation. This encourages the mask to be filled with ones, but allows some slack due to pixels that can not be explained by the photometric loss.

Figure 5.7: *This is an image extracted from the experiments in this thesis. The top image is frame $\mathbf{I}_t$ the middle image is frame $\mathbf{I}_{t-1}$ and the last image is the explain-ability mask. It is visible that the network correctly predicts that the bicycle is not moving in relation to the camera, but it does not remove pixels from the white van as it should.*

Stationary pixels mask The authors of [7] introduced a mask to remove stationary pixels from the set of previous, current and next frame. This is done by creating a mask where the photometric error is smaller before applying the projection than after. This works because pixels from objects that have not moved in relation to the camera will of course have a small photometric loss without reprojection. This will remove pixels from the car dashboard and also nearby vehicles that are traveling at the same speed, see Figure 5.8.

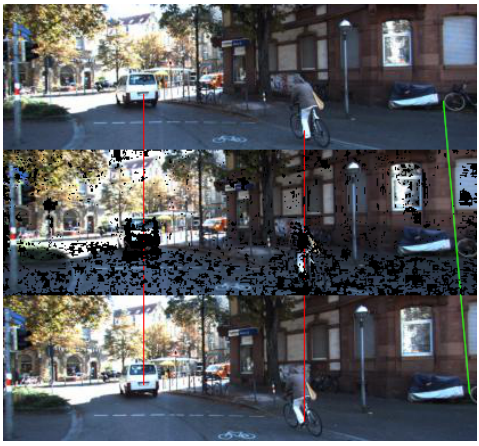

Figure 5.8: *The black pixels in the image are the ones removed because their photometric error is smaller before warping the image using the depth map compared to after warping. The red horizontal lines on the van and bicyclist illustrates that they do not move with respect to the camera, and the green slanted line illustrates that the bike leaning on the wall is moving with respect to the camera. The mask successfully removes pixels on the van and bicyclist, but also removes some pixels on the pavement that should not be removed.*

5.2 Unsupervised keypoint learning

The previous chapter we saw how neural networks can predict depth and ego motion. This can be seen as implicitly matching all the pixels in one frame with corresponding pixels in a nearby frame. But in order to build a map of the environment it is useful to extract sparse and repeatable features, or keypoints, from images. Every keypoints should have a unique descriptor which can be used to identify it in the map which makes it possible to expand the map and also localize the camera. How to build a map and localize in it is out of scope for this thesis. This section will describe the usnupervised learning method implemented in this thesis, to extract keypoints from images.

The method evaluated in this thesis is based on the UnsuperPoint paper[2]. The network architecture is illustrated in Figure 5.9. The input image is fed into a shared backbone. The features from the backbone are then split into three different encoders that estimate the score, position and a descriptor for each keypoint. The network will estimate a keypoint in every 8×8 patch of the image, but only the top N keypoints sorted by score are used in the evaluation.

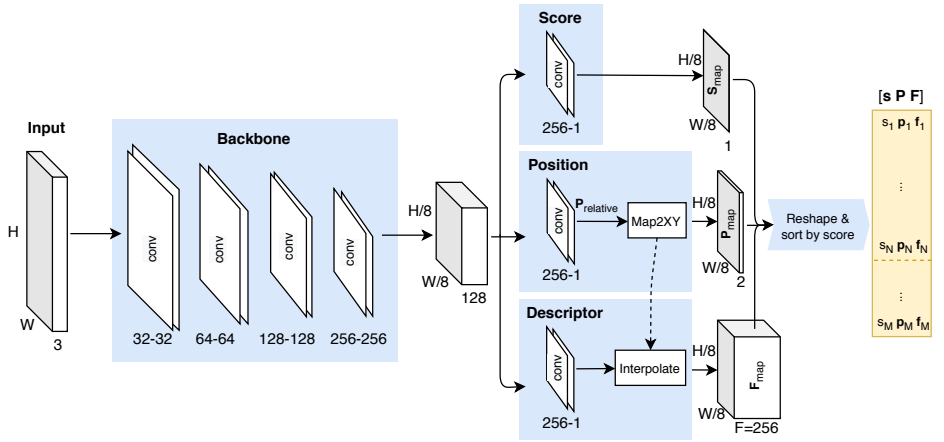

Figure 5.9: *The network has a common backbone and then splits into separate score, position and descriptor encoders. The output is reshaped and sorted by descending score.*

The score encoder is terminated by a sigmoid function and outputs $\mathbf{S}_{\text{map}}$, containing scores between 0 and 1 for each keypoint. The purpose of the scores are to rank the quality of the keypoints in all 8×8 px patches of the image. A subset of the keypoints with the highest scores are safe to use in the SFM system, while the keypoints with low scores are disregarded. Typically patches in the sky and other non-textured areas will have keypoints with low scores.

The position encoder is also terminated by a sigmoid function and outputs $\mathbf{P}_{\text{relative}}$ which is the relative position of the keypoint in each 8×8 patch. In the Map2XY block in Figure 5.9 the relative positions are converted to absolute pixel positions to form $\mathbf{P}_{\text{map}}$. The matrices have $H/8$ rows and $W/8$ columns, with two channels per cell storing the continuous x and y position for a predicted point in each patch. To convert from from relative to absolute point positions the following equations are used, where r is each row and c is each column in the matrices:

$$\mathbf{P}_{\text{map,x}}(r, c) = 8 * (c + \mathbf{P}_{\text{relative,x}}(r, c)) \tag{5.10}$$

$$\mathbf{P}_{\text{map,y}}(r, c) = 8 * (r + \mathbf{P}_{\text{relative,y}}(r, c)) \tag{5.11}$$

The descriptor encoder predicts a descriptor vector of length 256 for each keypoint. The purpose of the descriptor is to find corresponding keypoints in different images. The encoder produces 1 descriptor vector for each 8×8 patch in the image. This vector can be used directly as the keypoint descriptor and it works pretty well. Even better results can be achieved using the absolute keypoint positions to sample the values in the descriptor map with the same interpolation method used to do the warping in Figure 5.2.

$\mathbf{S}_{\text{map}}$, $\mathbf{P}_{\text{map}}$ and $\mathbf{F}_{\text{map}}$ are reshaped into a vector $\mathbf{s}$ of M elements, an $M \times 2$ matrix $\mathbf{P}$ and an $M \times 255$ matrix $\mathbf{F}$, where $M = \frac{W}{8} * \frac{H}{8} = 832$.

The network is trained in a siamese twin setup, where two duplicate networks that share weights are fed different inputs and the loss is formulated by comparing the output scores, positions and descriptors. The flow of data is illustrated in Figure 5.10.

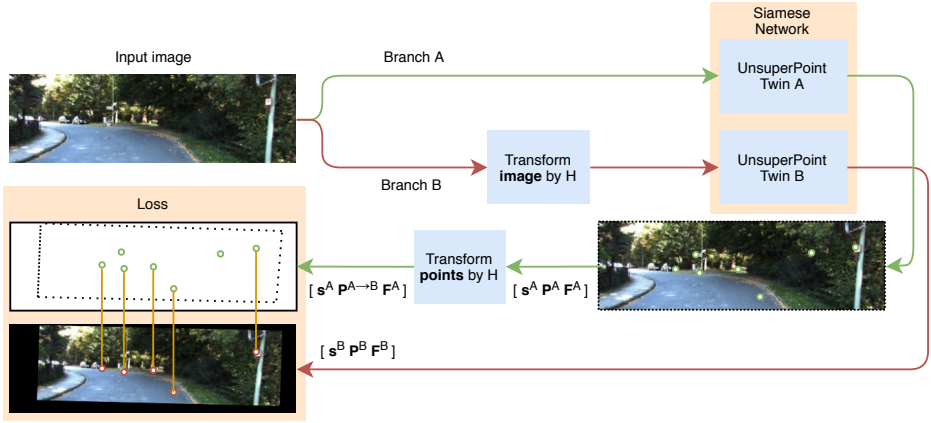

Figure 5.10: *This figure illustrates the flow from input image to loss function. The image from branch A is fed directly into the UnsuperPoint network, while in branch B the image is first transformed by a random homography* $\mathbf{H}$*. The keypoint positions from twin A are transformed by the same homography* $\mathbf{H}$ *and the output from the two branches are compared in order to formulate the loss function.*

For each branch $b \in \{A, B\}$ the siamese networks output the reshaped matrices $\mathbf{s}^b$, $\mathbf{P}^b$ and $\mathbf{F}^b$ which contain the scores, positions and descriptors of the M points in each image.

5.2.1 Loss function for keypoint learning

To formulate the loss function the point correspondences between branch A and B need to be determined. The points in branch A are transformed such that $\mathbf{p}_i^{A\to B} = \mathbf{H}\mathbf{p}_i^A$. Then an $M^A \times M^B$ distance matrix $\mathbf{G}$ is calculated from the pairwise distances between all points in each branch.

$$\mathbf{G} = [g_{ij}]_{M^A \times M^B} = \left[\|\mathbf{p}_i^{A\to B} - \mathbf{p}_j^B\|\right]_{M^A \times M^B} \tag{5.12}$$

We define a point pair if point i in branch A has a point j as its closest neighbor in branch B, and if the distance g_{ij} is less than 8px. With the point pairs defined the output matrices $\mathbf{s}^b$, $\mathbf{P}^b$ and $\mathbf{F}^b$ can be redefined as *corresponding matrices* $\hat{\mathbf{s}}^b$, $\hat{\mathbf{P}}^b$ and $\hat{\mathbf{F}}^b$ with $K \leq M$ entries, such that entry k in the new matrices maps to corresponding points in the input images.

Define d_k as the distance between each point pair.

$$d_k = \|\hat{\mathbf{p}}_k^{A \to B} - \hat{\mathbf{p}}_k^{B}\| \tag{5.13}$$

The distance between point pairs should be minimized, this is achieved by the $\mathcal{L}^{\text{position}}$ loss term.

$$\mathcal{L}_k^{\text{position}} = d_k \tag{5.14}$$

The scores of point pairs in branch A and B should be similar, this is achieved by the $\mathcal{L}^{\text{sim}}$ loss term.

$$\mathcal{L}_k^{\text{sim}} = \left(\hat{\mathbf{s}}_k^{A} - \hat{\mathbf{s}}_k^{B}\right)^2 \tag{5.15}$$

To teach the network to predict sensible scores for the points, the distance d_k between point pairs is used.

$$\mathcal{L}_k^{\text{score}} = \frac{\hat{\mathbf{s}}_k^{A} + \hat{\mathbf{s}}_k^{B}}{2}\left(d_k - \bar{d}\right) \tag{5.16}$$

If d_k is less than the mean distance $\bar{d}$ the score should be large in order to minimize the loss. If the d_k is greater than the mean distance $\bar{d}$ the score should be small in order to minimize the loss. Figure 5.11 shows the distribution of scores predicted by a fully trained network.

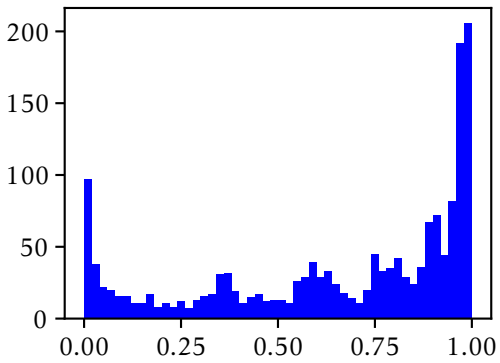

Figure 5.11: *Histogram with 50 buckets of scores for all points in a pair of images. The data is taken from the fully trained network evaluated in the results chapter. Most points has either a really low or high score, and a few points has a score somewhere in between.*

Training with only the above mentioned loss terms, the relative positions of the points in the 8×8 patches will be distributed towards the boundaries. One explanation for this is that hardly repeatable points will be encouraged to position them self near points in their neighboring patches, thereby minimizing d_k. The problem is illustrated in Figure 5.12

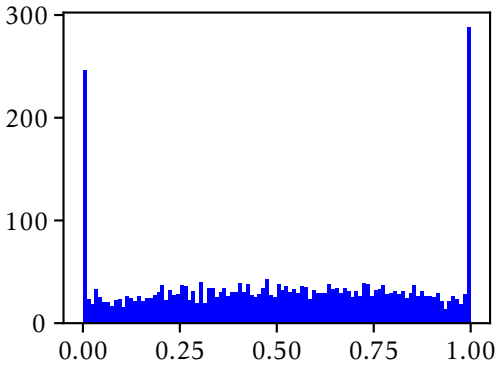

Figure 5.12: *Histogram with 100 buckets of relative point positions from both branch A and B, in both the x and y direction. As is clearly visible in the diagram there is a bias of relative positions towards 0 and 1.*

To alleviate this issue, a loss is added that encourages uniform distribution of the relative point positions.

$$\mathcal{L}^{\text{uniform}} = \mathcal{L}_A^{\text{uniform\_x}} + \mathcal{L}_A^{\text{uniform\_y}} + \mathcal{L}_B^{\text{uniform\_x}} + \mathcal{L}_B^{\text{uniform\_y}} \tag{5.17}$$

where

$$\mathcal{L}_A^{\text{uniform\_x}} = \sum_{i=1}^{M} \left(x_i^{\text{sorted}} - \frac{i-1}{M-1} \right)^2 \tag{5.18}$$

and so on for both branch A and B, in the x and y dimension. The effect of applying this loss can be seen in Figure 5.13.

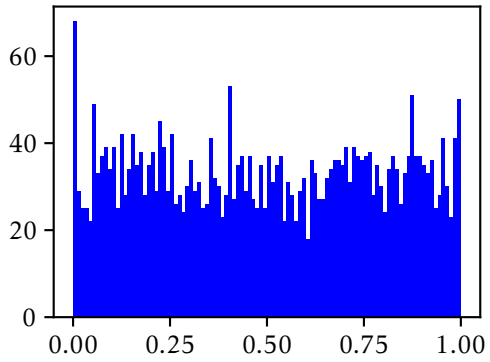

Figure 5.13: *Histogram with 100 buckets of relative point positions when including the loss term $\mathcal{L}^{\text{uniform}}$ that encourages a uniform distribution. The data is taken from the model evaluated in the results chapter of this thesis.*

The descriptor loss is calculated using a hinge loss with both positive margin m_p and negative margin m_n. An $M^A \times M^B$ correspondence matrix $\mathbf{C}$ is constructed, such that

$$c_{ij} = \begin{cases} 1 \text{ if } g_{ij} \leq 8 \\ 0 \text{ otherwise} \end{cases} \tag{5.19}$$

A maximum distance of 8px to classify two points as corresponding is suitable because of the patch size of 8×8 pixels. Unlike point pairs, a single point can correspond to multiple points in the other branch. The descriptor loss term is defined using the correspondance matrix $\mathbf{C}$ and descriptor matrix $\mathbf{F}$ as follows

$$\mathcal{L}^{\text{desc}} = \sum_{i=1}^{M^B} \sum_{j=1}^{M^B} \lambda_d * c_{ij} \max\left(0, m_p - \mathbf{f}_i^{A^T} \mathbf{f}_j^B\right) + (1 - c_{ij}) \max\left(0, \mathbf{f}_i^{A^T} \mathbf{f}_j^B - m_n\right) \tag{5.20}$$

Where $\mathbf{f}_i^b$ is row i of $\mathbf{F}^b$ for each branch $b \in \{A, B\}$. The weight λ_d is used to balance the few corresponding points compared to the non-corresponding ones. The descriptors are decorrelated by minimizing the off-diagonal elements of a descriptor correlation matrix $\mathbf{R}^b = [r_{ij}^b]_{F \times F}$.

$$\mathcal{L}^{\text{decorr}} = \sum_{i \neq j}^{\mathbf{R}} r_{ij}^A + \sum_{i \neq j}^{\mathbf{R}} r_{ij}^B \tag{5.21}$$

Each element r_{ij}^b for $b \in \{A, B\}$ is a Pearson's correlation coefficient[17] defined as

$$r_{ij}^b = \frac{\tilde{\mathbf{f}}_i^b \cdot \tilde{\mathbf{f}}_j^b}{\|\tilde{\mathbf{f}}_i^b\| \, \|\tilde{\mathbf{f}}_j^b\|}, \text{ where } \tilde{\mathbf{f}}_i^b = \mathbf{f}_i^b - \bar{\mathbf{f}}_i^b \text{ and } \tilde{\mathbf{f}}_j^b = \mathbf{f}_j^b - \bar{\mathbf{f}}_j^b \tag{5.22}$$

Where $\bar{\mathbf{f}}_i^b$ is the mean of vector $\mathbf{f}_i^b$. Due to what might be just a oversight by the authors, this is not the same definition of r_{ij} as presented in the original UnsuperPoint paper.

5.3 Unsupervised consensus maximization

Consensus maximization in a set of features related by a geometric constraint that also contains outliers is an important aspect of an SFM system. The problem consists of finding the largest set of inlier features that is consistent with a geometric constraint. In a 3D vision setting the geometric constraint could be for example a rigid transformation $[\mathbf{R}\ \mathbf{t}]$, a fundamental matrix $\mathbf{F}$ or a homography $\mathbf{H}$ that relates pairs of corresponding 3D or 2D points.

A popular algorithm to solve this problem is RANSAC[4], which is a classical iterative method. On the other hand, a unsupervised machine learning method for consensus maximization is presented in this paper[21]. The explanation of the method in the original paper uses very strict group theory. This section of the thesis will instead try to explain the method with emphasis on ease of understanding and not be as strict on the theory. The focus will be on explaining the method used to extract the parameters of the constraint, as it's not explained by the original paper in detail and took a lot of figuring out on my part, especially how to extract the parameters of the homography.

Consider one of the simplest geometric constraint on a set of corresponding 2D points in subsequent frames of the KITTI dataset, the fundamental matrix $\mathbf{F}$. Given a point $\mathbf{u}$ in frame 1, and point $\mathbf{v}$ in frame 2 they are related in the following way.

$$\mathbf{u}^T\mathbf{F}\mathbf{v} = 0 \tag{5.23}$$

To be more explicit the relation can be expressed as follows.

$$\begin{pmatrix} u_\mathrm{x} & u_\mathrm{y} & 1 \end{pmatrix} \begin{pmatrix} f_{11} & f_{12} & f_{13} \\ f_{21} & f_{22} & f_{23} \\ f_{31} & f_{32} & f_{33} \end{pmatrix} \begin{pmatrix} v_\mathrm{x} \\ v_\mathrm{y} \\ 1 \end{pmatrix} = 0 \tag{5.24}$$

Multiplication of the two rightmost matrices gives:

$$\begin{pmatrix} u_\mathrm{x} & u_\mathrm{y} & 1 \end{pmatrix} \begin{pmatrix} f_{11}v_\mathrm{x} + f_{12}v_\mathrm{y} + f_{13} \\ f_{21}v_\mathrm{x} + f_{22}v_\mathrm{y} + f_{23} \\ f_{31}v_\mathrm{x} + f_{32}v_\mathrm{y} + f_{33} \end{pmatrix} = 0 \tag{5.25}$$

Multiplication of the remaining two matrices and grouping $\mathbf{u}$ and $\mathbf{v}$ in parentheses for readability gives the following linear equation.

$$f_{11}(u_\mathrm{x}v_\mathrm{x}) + f_{12}(u_\mathrm{x}v_\mathrm{y}) + f_{13}(u_\mathrm{x}) + f_{21}(u_\mathrm{y}v_x) + f_{22}(u_\mathrm{y}v_\mathrm{y}) + f_{23}(u_\mathrm{y}) + f_{31}(v_\mathrm{x}) + f_{32}(v_\mathrm{y}) + f_{33}(1) = 0 \tag{5.26}$$

Now consider a set of m corresponding points that are related by the same fundamental matrix $\mathbf{F}$. Place the 9 monomials in the parentheses for all m points

in a matrix $\mathbf{M} \in \mathbb{R}^{mx9}$. A matrix of monomials in the form $1, a, a^2, \ldots$, such as this one, is called a Vandermonde matrix. A monomial is an expression in algebra that contains only one term.

$$\mathbf{M} = \begin{pmatrix} u_{x,1}v_{x,1} & u_{x,1}v_{y,1} & u_{x,1} & u_{y,1}v_{x,1} & u_{y,1}v_{y,1} & u_{y,1} & v_{x,1} & v_{y,1} & 1 \\ & & & \vdots & & & & & \\ u_{x,m}v_{x,m} & u_{x,m}v_{y,m} & u_{x,m} & u_{y,m}v_{x,m} & u_{y,m}v_{y,m} & u_{y,m} & v_{x,m} & v_{y,m} & 1 \end{pmatrix} \tag{5.27}$$

The constraint is satisfied by all m points if

$$\mathbf{M} \begin{pmatrix} f_{11} \\ f_{12} \\ f_{13} \\ f_{21} \\ f_{22} \\ f_{23} \\ f_{31} \\ f_{32} \\ f_{33} \end{pmatrix} = \begin{pmatrix} 0 \\ \vdots \\ 0 \end{pmatrix}^{mx1} \tag{5.28}$$

Now the solution $\mathbf{f}$ should look very familiar. The column vector $\mathbf{f}$ is in fact the very definition of a nullspace of $\mathbf{M}$. This is the key insight in this method of unsupervised consensus maximization. Finding the nullspace of $\mathbf{M}$ is done by performing a singular value decomposition, which is a differentiable operation.

$$\mathbf{U}\Sigma\mathbf{V}^T = \mathbf{M} \tag{5.29}$$

The nullspace will be the last column vector of $\mathbf{V} \in \mathbb{R}^{9x9}$ and the last singular value σ_9 in the diagonal of Σ will be zero. This is correct only if the set of corresponding points contains only inliers. If the set contains outliers that can not be related by the geometric constraint then the last singular value σ_9 will be greater than zero. If the inliers and outliers where known they could be represented by a vector $\mathbf{w} \in [0, 1]^m$ where 1 means inlier and 0 means outlier. Now $\mathbf{M}$ can be weighted by $\mathbf{w}$ which would cancel all rows in $\mathbf{M}$ which comes from an outlier pair of points.

$$\text{diag}(\mathbf{w})\mathbf{M}\mathbf{f} = \mathbf{0}$$

The problem of finding the largest set of inliers has now been transformed into finding the weights $\mathbf{w}$ such that the last singular value σ_9 of $\text{diag}(\mathbf{w})\mathbf{M}$ is near zero, and the sum of the weights should also be large to have as many inliers as possible.

To learn the weight $w_i \in \mathbf{w}$ for each pair of points, the PointNet[22] architecture was used. The pair of points are concatenated into rows of a $\mathbf{X} \in \mathbb{R}^{m \times n}$ matrix. For 2D points $n = 2 + 2 = 4$ and for 3D points $n = 3 + 3 = 6$. The input $\mathbf{X}$ is fed trough the PointNet segmentation network with a single output class,

signifying if the point pair is an inlier or not. The key innovation of the PointNet architecture is that it can directly consume point cloud data, as oppose to first transforming the input to a 3D voxel grid or an image. The network is also invariant to the ordering of the points in the input list. Figure 5.14 illustrates the process of feeding the input through pointnet to get the weights, and then applying the weights to the Vandermonde matrix of which we want to minimize the last singular value.

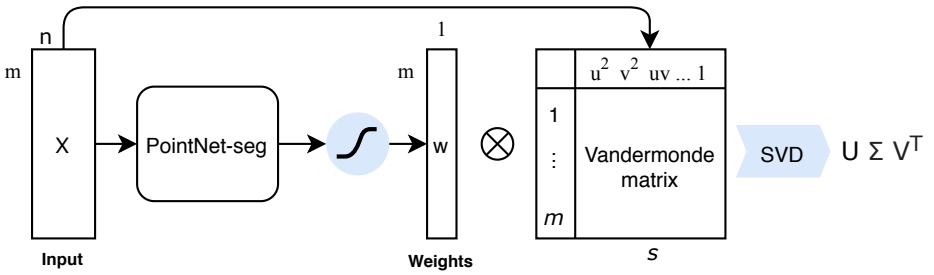

Figure 5.14: *The input point positions* $\mathbf{X} \in \mathbb{R}^{m \times n}$ *are fed trough the PointNet segmentation network, and are also used to construct the Vandermonde matrix* $\mathbf{M}$*. The rows of* $\mathbf{M}$ *are weighted using the weights* $\mathbf{w}$*. The network learns to predict inliers by minimizing the last singular values in* Σ *in the loss function.*

Because the fundamental matrix is a constraint expressed by 1 linear equation a basis of only 1 nullspace vector is needed from the singular value decomposition to construct it. But the method can also be used to learn constraints such as a homography or rigid 3D transformation that are constrained by 3 linear equations and therefore require a basis of 3 nullspace vectors. In that case the last 3 singular values from the SVD should be minimized in the loss function.

In the following sections the method used to extract the homography from the basis vectors will be explained. Two points that are related by a homography is expressed as follows. The monomials are grouped by parentheses.

$$\mathbf{u} \times \mathbf{H}\mathbf{v} = 0 \rightarrow \tag{5.30}$$

$$[\mathbf{u}]_{\times}\mathbf{H}\mathbf{v} = 0 \rightarrow \tag{5.31}$$

$$\begin{pmatrix} 0 & -1 & u_y \\ 1 & 0 & -u_x \\ -u_y & u_x & 0 \end{pmatrix} \begin{pmatrix} h_{11} & h_{12} & h_{13} \\ h_{21} & h_{22} & h_{23} \\ h_{31} & h_{32} & h_{33} \end{pmatrix} \begin{pmatrix} v_x \\ v_y \\ 1 \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \\ 0 \end{pmatrix} \rightarrow \tag{5.32}$$

$$-h_{21}(v_x) \quad -h_{22}(v_y) \quad -h_{23}(1) \quad +h_{31}(v_x u_y) + h_{32}(v_y u_y) + h_{33}(u_y) = 0 \tag{5.33}$$

$$h_{11}(v_x) \quad +h_{12}(v_y) \quad +h_{13}(1) \quad -h_{31}(v_x u_x) - h_{32}(v_y u_x) - h_{33}(u_x) = 0 \tag{5.34}$$

$$-h_{11}(v_x u_y) - h_{12}(v_y u_y) - h_{13}(u_y) + h_{21}(v_x u_x) + h_{22}(v_y u_x) + h_{23}(u_x) = 0 \tag{5.35}$$

The monomials in the parentaccess are the same as for the fundamental constraint, which means that the same Vandermonde matrix $\mathbf{M}$ can be used. But because an homography is constrained by $r = 3$ linear equations the last 3 singular values of the SVD should be minimized during training. Extract 3 basis vectors from the nullspace which will be the 3 rightmost columns of $\mathbf{V}$ in the SVD of diag($\mathbf{w}$)$\mathbf{M}$.

$$\mathbf{B} = \begin{pmatrix} \mathbf{v}_7 & \mathbf{v}_8 & \mathbf{v}_9 \end{pmatrix} \in \mathbb{R}^{9x3} \tag{5.36}$$

When calculating the fundamental matrix, the nullspace of the vandermonde matrix had a direct one to one mapping with the parameters needed to construct $\mathbf{F}$. This is no longer the case when we want to use the nullspace basis $\mathbf{B}$ to construct the homography matrix $\mathbf{H}$. This is because the 3 linear equations of H are tangled. Notice that h_{21}, h_{22} and h_{23} appears both in equation 5.33 and 5.35. Also h_{31}, h_{32} and h_{33} appears both in equation 5.33 and 5.35. Also h_{11}, h_{12} and h_{13} appears both in equation 5.34 and 5.35. In order to extract the elements of $\mathbf{H}$ from the nullspace basis vectors in $\mathbf{B}$ we need to perform a change of basis. Notice that equation 5.33 does not contain u_x and equation 5.34 does not contain u_y. We can exploit this fact to perform a change of basis on $\mathbf{B} \in \mathbb{R}^{9x3}$ to get $\mathbf{B}' \in \mathbb{R}^{9x2}$ that should have the structure seen in Table 5.1.

Monomial	Structure of $\mathbf{B}'$	Corresponding element in $\mathbf{H}$
$u_x v_x$	$\begin{pmatrix} 0 & . \end{pmatrix}$	$\begin{pmatrix} & h_{31} \end{pmatrix}$
$u_x v_y$	$\begin{pmatrix} 0 & . \end{pmatrix}$	$\begin{pmatrix} & h_{32} \end{pmatrix}$
u_x	$\begin{pmatrix} 0 & . \end{pmatrix}$	$\begin{pmatrix} & h_{33} \end{pmatrix}$
$u_y v_x$	$\begin{pmatrix} . & 0 \end{pmatrix}$	$\begin{pmatrix} h_{31} & \end{pmatrix}$
$u_y v_y$	$\begin{pmatrix} . & 0 \end{pmatrix}$	$\begin{pmatrix} h_{32} & \end{pmatrix}$
u_y	$\begin{pmatrix} . & 0 \end{pmatrix}$	$\begin{pmatrix} h_{33} & \end{pmatrix}$
v_x	$\begin{pmatrix} . & . \end{pmatrix}$	$\begin{pmatrix} h_{21} & h_{11} \end{pmatrix}$
v_y	$\begin{pmatrix} . & . \end{pmatrix}$	$\begin{pmatrix} h_{22} & h_{12} \end{pmatrix}$
1	$\begin{pmatrix} . & . \end{pmatrix}$	$\begin{pmatrix} h_{23} & h_{13} \end{pmatrix}$

Table 5.1: *The basis monomials, the structure of $\mathbf{B}'$ in the new basis and its corresponding elements in $\mathbf{H}$.*

To perform the change of basis to get the structure of $\mathbf{B}'$ we use the nullspace $\mathbf{n}_1 \in \mathbb{R}^{3\times1}$ of rows 1, 2 and 3 from $\mathbf{B}$, and the nullspace $\mathbf{n}_2 \in \mathbb{R}^{3x1}$ of row 4, 5, 6 from $\mathbf{B}$.

$$\mathbf{A}_1 = \begin{pmatrix} b_{11} & b_{12} & b_{13} \\ b_{21} & b_{22} & b_{23} \\ b_{31} & b_{32} & b_{33} \end{pmatrix}, \mathbf{A}_2 = \begin{pmatrix} b_{41} & b_{42} & b_{43} \\ b_{51} & b_{52} & b_{53} \\ b_{61} & b_{62} & b_{63} \end{pmatrix} \tag{5.37}$$

$$\mathbf{n}_1 = \text{"}\textit{rightmost column of right-singular vectors of } \mathbf{A}_1\text{"} \tag{5.38}$$

$$\mathbf{n}_2 = \text{"}\textit{rightmost column of right-singular vectors of } \mathbf{A}_2\text{"} \tag{5.39}$$

$$\mathbf{b}^{\mathbf{n}_1} = \mathbf{B}\mathbf{n}_1 \tag{5.40}$$

$$\mathbf{b}^{\mathbf{n}_2} = \mathbf{B}\mathbf{n}_2 \tag{5.41}$$

Now the $\mathbf{b}^{\mathbf{n}_1}$ and $\mathbf{b}^{\mathbf{n}_2}$ basis vectors will have the following structure as desired.

$$\mathbf{b}^{\mathbf{n}_1} = \begin{pmatrix} 0 \\ 0 \\ 0 \\ . \\ . \\ . \\ . \\ . \\ . \end{pmatrix}, \mathbf{b}^{\mathbf{n}_2} = \begin{pmatrix} . \\ . \\ . \\ 0 \\ 0 \\ 0 \\ . \\ . \\ . \end{pmatrix}, \tag{5.42}$$

The basis vectors have zeros at the correct place. The next step is to adjust the scale so that row 4, 5 and 6 of $\mathbf{b}^{\mathbf{n}_1}$ and row 1, 2 and 3 of $\mathbf{b}^{\mathbf{n}_2}$ have the same norm because they both represent the same elements h_{31}, h_{32} and h_{33}. We also make sure they have the same sign.

$$s = \text{sign}(b_1^{\mathbf{n}_2} + b_2^{\mathbf{n}_2} + b_3^{\mathbf{n}_2})\,\text{sign}(b_4^{\mathbf{n}_1} + b_5^{\mathbf{n}_1} + b_6^{\mathbf{n}_1}) \tag{5.43}$$

s will be -1 if they have different signs, or 1 if they are the same sign.

$$\mathbf{B}' = \left(\mathbf{b}^{\mathbf{n}_1} / \left\|(b_4^{\mathbf{n}_1}\ b_5^{\mathbf{n}_1}\ b_6^{\mathbf{n}_1})\right\| \quad \mathbf{b}^{\mathbf{n}_2} / \left\|(b_1^{\mathbf{n}_2}\ b_2^{\mathbf{n}_2}\ b_3^{\mathbf{n}_2})\right\| s\right) \in \mathbb{R}^{9x2} \tag{5.44}$$

It is now possible to assign the elements of $\mathbf{B}'$ to the corresponding element of $\mathbf{H}$ (Table 5.1) in the following way. The last row is negated because h_{31}, h_{32} and h_{33} has a different signs in equation 5.33 compared to 5.34.

$$\mathbf{H} = \begin{pmatrix} b'_{72} & b'_{82} & b'_{92} \\ b'_{71} & b'_{81} & b'_{91} \\ -b'_{41} & -b'_{51} & -b'_{61} \end{pmatrix} \tag{5.45}$$

The method not only works for predicting the fundamental or homographic relationship between points, but can also be used for 3D rigid transformations.

$$\mathbf{v} = \mathbf{R}\mathbf{u} + \mathbf{t} \rightarrow \tag{5.46}$$

$$\mathbf{R}\mathbf{u} + \mathbf{t} - \mathbf{v} = \mathbf{0} \rightarrow \tag{5.47}$$

$$\begin{pmatrix} r_{11} & r_{12} & r_{13} \\ r_{21} & r_{22} & r_{23} \\ r_{31} & r_{32} & r_{33} \end{pmatrix} \begin{pmatrix} u_x \\ u_y \\ y_z \end{pmatrix} + \begin{pmatrix} t_x \\ t_y \\ t_z \end{pmatrix} - \begin{pmatrix} v_x \\ v_y \\ v_z \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \\ 0 \end{pmatrix} \rightarrow \tag{5.48}$$

$$\begin{cases} r_{11}(u_x) + r_{12}(u_y) + r_{13}(u_z) + t_x(1) - (v_y) = 0 \\ r_{21}(u_x) + r_{22}(u_y) + r_{23}(u_z) + t_y(1) - (v_x) = 0 \\ r_{31}(u_x) + r_{32}(u_y) + r_{33}(u_z) + t_z(1) - (v_z) = 0 \end{cases} \tag{5.49}$$

The monomials in the parentheses form the following Vandermonde matrix.

$$\mathbf{M} = \begin{pmatrix} u_{\mathrm{x},1} & u_{\mathrm{y},1} & u_{\mathrm{z},1} & v_{\mathrm{x},1} & v_{\mathrm{y},1} & v_{\mathrm{z},1} & 1 \\ & & & \vdots & & & \\ u_{\mathrm{x},m} & u_{\mathrm{y},m} & u_{\mathrm{z},m} & v_{\mathrm{x},m} & v_{\mathrm{y},m} & v_{\mathrm{z},m} & 1 \end{pmatrix} \tag{5.50}$$

The constraint of $r = 3$ linear equations is satisfied for all inliers if

$$\mathrm{diag}(\mathbf{w})\mathbf{M} \begin{pmatrix} r_{11} & r_{21} & r_{31} \\ r_{12} & r_{22} & r_{32} \\ r_{13} & r_{23} & r_{33} \\ -1 & 0 & 0 \\ 0 & -1 & 0 \\ 0 & 0 & -1 \\ t_{\mathrm{x}} & t_{\mathrm{y}} & t_{\mathrm{z}} \end{pmatrix} = \begin{pmatrix} 0 \\ \vdots \\ 0 \end{pmatrix}^{mx1} \tag{5.51}$$

Similar to before $\mathbf{R}$ and $\mathbf{t}$ is extracted from the null vectors of $\mathrm{diag}(\mathbf{w})\mathbf{M}$. The null vectors are the 3 rightmost singular vectors in the SVD. The 3 null vectors form a basis matrix $\mathbf{B}$ on which a change of basis is performed to untangle the components v_{x}, v_{y} and v_{z} into a new structure $\mathbf{B}'$ where $\mathbf{R}$ and $\mathbf{t}$ are easy to extract.

Monomial	Structure of $\mathbf{B}'$
u_x	$\begin{pmatrix} r_{11} & r_{21} & r_{31} \end{pmatrix}$
u_{y}	$\begin{pmatrix} r_{12} & r_{22} & r_{32} \end{pmatrix}$
u_{z}	$\begin{pmatrix} r_{13} & r_{23} & r_{33} \end{pmatrix}$
v_{x}	$\begin{pmatrix} -1 & 0 & 0 \end{pmatrix}$
v_{y}	$\begin{pmatrix} 0 & -1 & 0 \end{pmatrix}$
v_{z}	$\begin{pmatrix} 0 & 0 & -1 \end{pmatrix}$
1	$\begin{pmatrix} t_{\mathrm{x}} & t_{\mathrm{y}} & t_{\mathrm{z}} \end{pmatrix}$

$$\mathbf{B}' = -\begin{pmatrix} b_{41} & b_{42} & b_{43} \\ b_{51} & b_{52} & b_{53} \\ b_{61} & b_{62} & b_{63} \end{pmatrix}^{-1} \begin{pmatrix} b_{11} & b_{12} & b_{13} \\ b_{21} & b_{22} & b_{23} \\ b_{31} & b_{32} & b_{33} \\ b_{41} & b_{42} & b_{43} \\ b_{51} & b_{52} & b_{53} \\ b_{61} & b_{62} & b_{63} \\ b_{71} & b_{72} & b_{73} \end{pmatrix} = \begin{pmatrix} r_{11} & r_{21} & r_{31} \\ r_{12} & r_{22} & r_{32} \\ r_{13} & r_{23} & r_{33} \\ -1 & 0 & 0 \\ 0 & -1 & 0 \\ 0 & 0 & -1 \\ t_{\mathrm{x}} & t_{\mathrm{y}} & t_{\mathrm{z}} \end{pmatrix} \tag{5.52}$$

However this holds for any affine transformation, to enforce the rotation manifold constraint an additional regularizer loss term is added.

$$\mathcal{L}_r = \log(\mathbf{1} + \|\mathbf{R}\mathbf{R}^T - \mathbf{I}_{3\times 3}\|)$$

In the confusion matrix in the results section a point is classified as an inlier if its weight $w_i > 0.5$. An example of the distribution of the weights w_i for a typical set of input points can be seen in Figure 5.15.

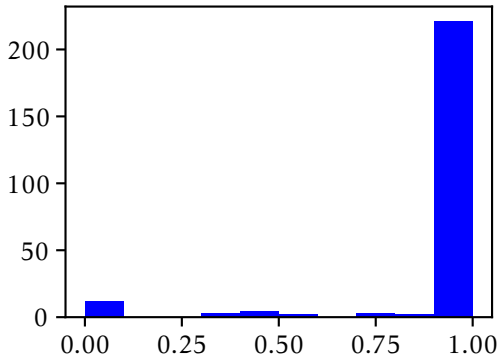

Figure 5.15: *Histogram with 10 buckets of inlier weights w_i for all point correspondences in a pair of images. A point is classified as an inlier if the weight w_i is above a certain threshold.*

5.3.1 Improving training convergence

After some initial attempts at training the consensus maximization network for homographies on the output of the keypoint network, it was clear that it was next to impossible to get the training to converge on a good solution. Applying some additional techniques not described in the original paper resolved the issue.

Firstly the points should be transformed into a different basis before they are fed into the consensus maximization network. The change of basis ensures that the coordinates are scaled so that their maximum magnitude is 1, and the origin is centered in the middle of the image and not in the top left corner.

$$\mathbf{G} = \begin{pmatrix} W & 0 & W/2 \\ 0 & W & H/2 \\ 0 & 0 & 1 \end{pmatrix} \tag{5.53}$$

$$\mathbf{p}' = \mathbf{G}^{-1}\mathbf{p} \tag{5.54}$$

Where $\mathbf{p}$ is the output point from the keypoint network, W and H is the width and height of the image respectively, and $\mathbf{p}'$ is the new altered point used as input. The homography $\mathbf{H}$ predicted by the consensus maximization network will be in this new basis $\mathbf{G}$ and needs to be altered in order to use it in our standard pixel coordinate basis as follows.

$$\mathbf{H}' = \mathbf{G}\mathbf{H}\mathbf{G}^{-1} \tag{5.55}$$

The second method to improve convergence during training is to normalize the rows of the Vandermonde matrix as follows.

$$\mathbf{M}'_n = \frac{\mathbf{M}_n}{||\mathbf{M}_n||} \tag{5.56}$$

For all n rows of the Vandermonde matrix $\mathbf{M}$.

6
Results

This chapter presents the results from the experiments conducted to evaluate the performance of the unsupervised learning methods investigated in this thesis. The chapter is structured into three sections with results from the depth and ego motion prediction, keypoint prediction and consensus maximization respectively. The metrics used to evaluate the networks are detailed in the evaluation metrics section 4.4 in the method chapter.

6.1 Depth and ego motion

For the depth and ego motion prediction networks 12 different configurations (Table 6.1) where trained and evaluated in 16 experiments (Table 6.2). The configurations and experiments are named C1 trough C12 and E1 to E16 respectively. A configuration is a particular combination of options (Table 4.2), such as dataset, network architecture and loss function terms used during training. Each experiment measures the performance, using the metrics in section 4.4.1, of a configuration on a particular testing dataset. In some experiments the testing dataset will differ from the training dataset, for example a network can be trained on Kitti but evaluated on Lyft.

C	Net	DS	Edge	Norm	Expl	Stat	SSIM	Comb	US
1	SL	K						avg	
2	SL	K			×			avg	
3	SL	K				×		avg	
4	SL	K	×			×		avg	
5	SL	K	×			×	×	min	
6	SL	K	×	×		×	×	min	
7	SL	K	×	×		×	×	min	×
8	SL	L	×	×		×	×	min	×
9	M2	K				×		avg	
10	M2	K	×	×		×	×	min	
11	M2	K	×	×		×	×	min	×
12	M2	L	×	×		×	×	min	×

Table 6.1: *All configurations of the depth and ego motion network that were evaluated. The C column identifies the specific configuration. The Net column shows which architecture was used, either SL for SfmLearner or M2 for Monodepth2. The DS column shows which dataset was used during training, either K for Kitti or L for Lyft. The Edge column indicates if the edge aware smoothing loss term $\mathcal{L}_{edge}$ was used. The Norm column indicates if depth map normalization was used. The Expl column indicates if an explainability mask was used. The Stat column indicates if a mask to remove stationary pixels from the loss was used. The SSIM column indicates if $\mathcal{L}_{ps}$ was used, otherwise just $\mathcal{L}_{p}$. The Comb column shows which methods was used to combine the loss terms from the two source images, either the average or the minimum loss across frames. The US column indicates that up scaling of the depth maps in the pyramid was used, otherwise the target frame was down scaled to match the size of the smaller depth maps in the pyramid.*

E	C	DS	AbsRel	SqRel	RMSE	RMSLE	1.25	1.25^2	1.25^3	Ego
1	1	K	0.174	1.405	4.829	0.249	0.784	0.920	0.964	0.024
2	2	K	0.179	1.686	4.880	0.252	0.782	0.919	0.961	0.021
3	3	K	0.140	0.793	4.549	0.217	0.818	0.939	0.976	0.020
4	4	K	0.143	0.819	4.708	0.222	0.810	0.937	0.975	0.022
5	5	K	0.133	0.727	4.305	0.204	0.843	0.950	0.979	0.023
6	6	K	0.137	0.797	4.282	0.208	0.837	0.948	0.977	0.024
7	7	K	0.135	0.778	4.248	0.208	0.841	0.948	0.997	0.020
8	7	L	0.340	7.811	23.071	0.447	0.457	0.734	0.868	0.043
9	8	K	0.512	5.185	10.757	0.611	0.305	0.549	0.732	0.495
10	8	L	0.739	20.236	31.859	0.773	0.240	0.434	0.587	1.324
11	9	K	**0.125**	**0.697**	4.298	0.203	0.845	0.948	0.979	0.021
12	10	K	0.126	0.714	4.018	**0.194**	**0.860**	**0.958**	**0.982**	**0.019**
13	11	K	0.132	0.769	**3.966**	0.196	0.859	0.957	0.981	**0.019**
14	11	L	0.304	7.019	21.907	0.414	0.518	0.775	0.886	0.042
15	12	K	0.322	3.177	7.179	0.378	0.549	0.797	0.906	0.036
16	12	L	0.303	8.051	19.312	0.385	0.637	0.827	0.906	0.059

Table 6.2: *All the experiments measuring the performance of the different configurations in Table 6.1. The E column identifies a specific experiment. The C column shows which configuration was used. The DS column shows which dataset was used during testing, K for Kitti and L for Lyft. The dataset used during testing differs from the one used during training in some experiments. The AbsRel, SqRel, RMSE and RMSLE columns are the depth error metrics described in section 4.4.1, smaller is better. The* 1.25, 1.25^2 *and* 1.25^3 *columns are the depth accuracy metrics described in section 4.4.1, larger is better. The Ego column is the camera ego motion error metric described in section 4.4.2.*

To verify the quality of the predictions visually the depth map can be rendered using two different methods. Either as tinted depth map where black pixels are further away than yellow pixels (Figure 6.1), or as a 3D point cloud where each point is colored by the input image (Figure 6.2).

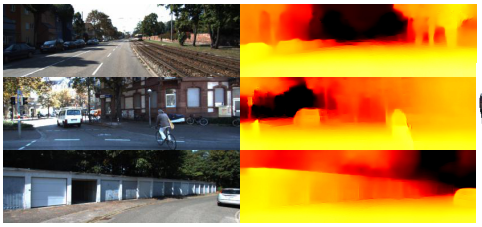

Figure 6.1: Three depth maps from experiment E13, which is the Monodepth2 architecture with all options enabled, trained and evaluated on the Kitty dataset.

Figure 6.2: One depth map from experiment E13 rendered as a colorized point cloud.

The point cloud of a particular image can be rendered from different perspectives by a virtual camera (Figure 6.3). The virtual camera can be used to render a birds eye view of the 3D path built using the incremental ego motion predictions in a sequence (Figure 6.4).

Figure 6.3: A single depth map from experiment E13 rendered from 3 different angles as a colorized point cloud.

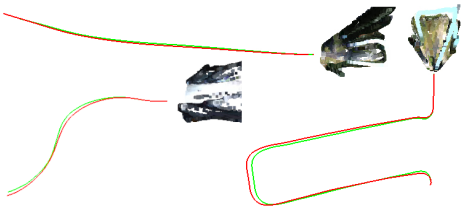

Figure 6.4: Camera movement in three different image sequences from experiment E13. The green lines are the ground truth and the red lines are the predicted camera trajectories.

Comparing E1 and E2, we see that the explainability mask improves the accuracy ever so slightly but the depth error is actually worse. When the stationary pixel mask is used instead in E3 the error metrics are significantly better than in both E1 and E2, the accuracy is comparable but slightly worse. The results correspond to what can be seen visually in Figure 6.5.

In experiment E4 the edge aware depth smoothness term is added, interestingly the performance metric does not show an improvement. But looking at one of the depth maps (Figure 6.6) it visually looks significantly more sharp and well defined. Probably the depth error moves to other parts of the image when the edge aware depth smoothness term is enabled.

In experiment E5 we use SSIM for the photometric reconstruction error, and also combine the loss from $\mathbf{I}_{t-1}$ and $\mathbf{I}_{t+1}$ using the min() function per pixel. All metrics show an improvement.

In E6 we add depth map normalization and in E7 we add upscaling, neither show any significant changes in the metrics.

In E8 we test configuration C7 on the Lyft dataset, though it was trained on Kitti. As expected, the results are worse when evaluated on a completely different dataset than the model was trained on.

In experiment E9 and E10 configuration C8 trained on Lyft is evaluated on Kitty and Lyft respectively. Very surprisingly the model performs better on Kitti even though it was trained on Lyft.

From experiment E11 and onwards the SfMLearner architecture is replaced by the Monodepth2 architecture. Both E11 and E12 shows great performance, even though E11 has a configuration which is much more bare bones than E12. This suggests that the network architecture plays a big part in the overall performance.

In E13 and E14 we again see that a network trained on Kitti performs better when evaluated on Kitti compared to Lyft, as expected. In E15 and E16 we see a network trained on Lyft that has a lower error on Kitti, but higher accuracy on Lyft.

From the results it appears that the quality of depth prediction has a pretty low impact on the performance of the ego motion estimation. The depth and ego motion is predicted by different networks. But the loss function is dependent on both depth and ego motion predictions. It's not possible to learn only depth or only ego motion separately in this system, so they are in a way coupled. But the networks are separate and the performance of the depth network does not seem to have a big impact on the performance of the ego motion network.

The effect of using either a explainability mask or stationary pixel mask can be seen in Figure 6.5.

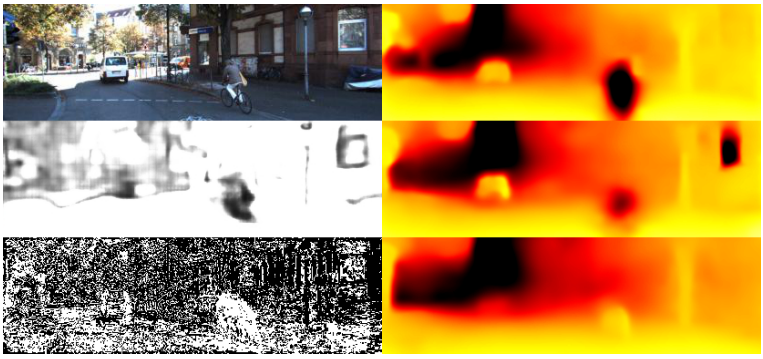

Figure 6.5: *The top row is from experiment E1, the second row is from E2 with the explainability maps shown to the left, and the last row is from E3 with the stationary pixel map shown to the left. In the top depth map the depth prediction for the biker is incorrect. Because the biker is stationary with respect to the camera the disparity of the pixels across the subsequent frames becomes 0 and the depth goes to infinity. Training using an explainability mask seems to improve the depth prediction for the biker, and using a stationary pixels mask improves the depth even more.*

Figure 6.6: *A depth map taken from experiment E4 where the edge aware smoothness loss is used.*

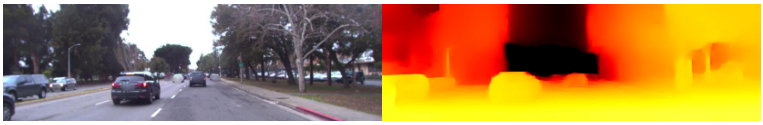

Figure 6.7: *Depth map from experiment E14, where the model was trained on Kitti and evaluated on Lyft. As can be seen in the image the performance is still quite good.*

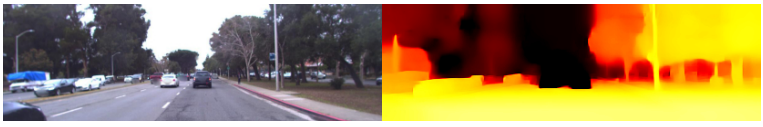

Figure 6.8: *Depth map from experiment E15 where the model was trained on Lyft and also evaluated on Lyft.*

6.2 Keypoint detection

The keypoint predicting network was evaluated and compared with the ORB detector in OpenCV using the metrics described in section 4.4.3. An example of the output of the keypoint prediction network can be seen in Figure 6.9.

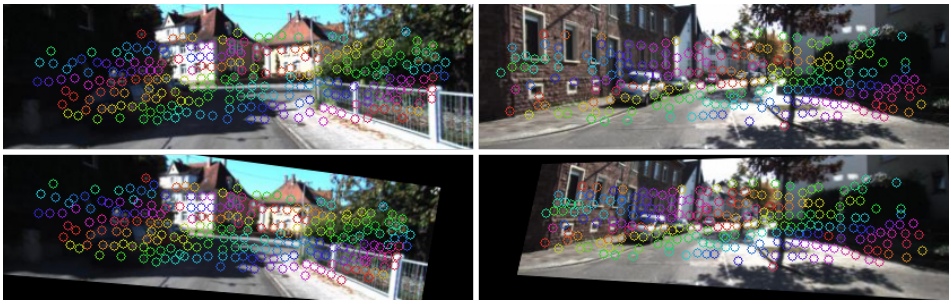

Figure 6.9: *Results from the keypoint prediction network. The top row are the original input images fed to branch A, and bellow are the transformed images fed to branch B. Circles that are the same color have matching descriptors.*

Method	RS ↑	LE ↓	MS ↑	MR ↑
UnsuperPoint	0.796	**0.666**	**0.488**	**0.834**
ORB	**0.841**	0.764	0.302	0.564

Table 6.3: *Comparison between the keypoint predicting network and ORB detector in OpenCV. The table shows the repeatability score, localization error, matching score and matching ratio.*

As can be seen in Table 6.3 the repeatability score is better for the ORB detector. This means that the ORB detector is picking points in different frames that are likely to represent the same physical feature in the world. On the other hand the keypoint network trained in this thesis has a lower localization error. The matching score and matching ratio is also better for the keypoint network, which suggests that the descriptors are better than its ORB counterpart.

6.3 Consensus maximization

The consensus maximization network was evaluated and compared with the *findHomography()* function in OpenCV using the metrics described in section 4.4.4. An example the results from the consensus maximization network can be seen in Figure 6.10.

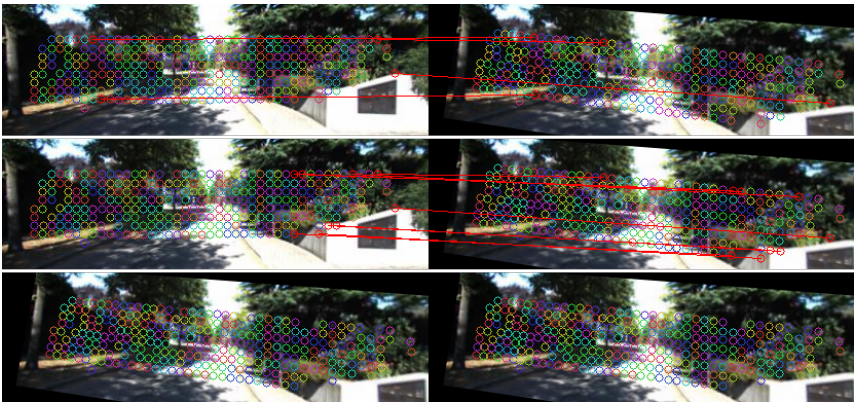

Figure 6.10: *Outlier prediction on homographic adaptation dataset with images from Kitti. First row shows the outliers found by OpenCV findHomography() marked in red. Second row shows the outliers found by the consensus network marked in red. In the third row the image on the left is is transformed by the homography predicted by the consensus network, and the image to the right is the image transformed by the ground truth homography.*

Method	HE (std) ↓	Best 80% HE (std) ↓
ConsensusNet	3.97 (22.27)	1.74 (0.82)
RANSAC	0.61 (0.45)	0.45 (0.12)

Table 6.4: *Comparison between the consensus maximization network and findHomography() function of OpenCV which uses the RANSAC algorithm.*

As can be seen in Table 6.4 the homography error (HE section 4.4.4) is larger for the consensus network compared to using RANSAC. The standard deviation is very high, which suggests that the average error contains large outliers. Filtering out the best 80% of the testing samples, we see that the standard deviation of the average homography error drops significantly. Even though the homography error is still worse than compared with RANSAC, it is not far off.

		Predicted		
		Inlier	Outlier	Total
Actual	Inlier	0.964	0.036	TP + FN = 1
	Outlier	0.002	0.998	FP + TN = 1
	Total	TP + FP = 0.966	FN + TN = 1.034	2

Table 6.5: *Confusion matrix for inlier/outlier prediction using ConsensusNet.*

		Predicted		
		Inlier	Outlier	Total
Actual	Inlier	0.983	0.017	TP + FN = 1
	Outlier	0.196	0.804	FP + TN = 1
	Total	TP + FP = 1.179	FN + TN = 0.821	2

Table 6.6: *Confusion matrix for inlier/outlier prediction using RANSAC.*

Looking at Table 6.5 and Table 6.6 we can observe that the consensus network is biased towards predicting more outliers, while RANSAC is biased towards predicting more inliers. Notice that the consensus network is biased towards outliers while still having a worse homography error compared to RANSAC. Overall the performance of both the outlier prediction and homography estimation aspect of the consensus maximization network is acceptable.

7

Discussion

This chapter aims to discuss and critique the work that has been presented in this thesis.

7.1 Results

In this section we discuss and draw conclusions from the experiments.

7.1.1 Depth and ego motion

From the experiments it appears that most options evaluated to alter the loss function does help to improve the results. Looking at experiment E7 where a maxed out SfMLearner architecture was trained and comparing it with experiment E11 where a very bare bones Monodepth2 architecture was trained we see that they have almost the same performance. This suggests that even though a good loss function is important, perhaps even more important is the underlying network architecture.

I am hesitant to draw any conclusions from experiment E9 and E10 where a model trained on Lyft performs better on Kitti than on the dataset it was trained on. This is so unexpected that I think it is more likely that the testing ground truth data is corrupt, but I have been unable to find any obvious problems with the data upon visual inspection.

Another surprise was the negligible effect of adding upscaling to the smaller depth maps. In the original paper it was claimed to be quite effective, but I was not able to come to the same conclusion.

It is quite clear from the results that the stationary pixel mask is more effective than the explainability mask. I also think there are further improvements that can be made here, as the stationary pixel mask is quite noisy.

Comparing experiment E3 and E4 we see that the edge aware depth smoothness loss does not help to improve the evaluation metrics. I still think it is valuable technique because the depth map looks sharper by visual inspection. Comparing experiment E4 and E5 we see that using SSIM in the photometric error and combining the per pixel loss across frames using the min function improves the metrics significantly.

7.1.2 Keypoint detection

The results from the keypoint prediction network was very satisfactory. One interesting property of the predicting network is that it rarely predicts keypoints at the edges of the image. I believe this might be because points near the edges in branch A are often outside of the image in branch A due to the homographic transformation. This means that predicting points near the edges would be penalized in the loss function during training because no correspondence would be found between points in branch A and B.

7.1.3 Consensus maximization

The results from the consensus maximization network is promising. But it seems to be sensitive to a few samples in the test set that throws it off completely thereby causing a large standard deviation in the mean homographic error (HE).

7.2 Method

This section aims to critique and discuss the method used to answer our research questions.

7.2.1 Datasets

Training the depth predicting network on Kitti and testing it on Lyft we can see that the depth maps produced (Figure 6.7) is quite acceptable. It is important for real world applications that the network is not over-fitted to its training dataset, but instead generalizes well to unseen data. Training and evaluating on two rather different car sequence dataset in this thesis was valuable in this regard, many papers only focus on getting the best benchmark on Kitti.

Using images from the Kitti dataset to train the keypoint prediction network gave the insight that the technique generalizes well to the domain of autonomous cars and navigation. In the original paper the network was trained on the HPatches[1] dataset. A drawback of using homographic transformation to train the keypoint network is that is it unclear if the keypoint network would generalize to images not related by a simple homographic transformation. In non planar scenes where the camera is moving in 3D the feature points would instead be related by a fundamental matrix. To setup a system where unsupervised training could be performed on an image sequance such as the Kitti sequance dataset would be more complicated, and would be an interesting topic for future work.

The consensus maximization network was trained on homographies because it was possible to implement it as a natural extension of the keypoint prediction pipeline. If the keypoint prediction training pipeline is altered to train on an image sequence, then the consensus maximization could predict a fundamental matrix instead of an homography, which also would be more appropriate to solve a structure from motion problem.

7.2.2 Evaluation metrics

Comparing the keypoint prediction network with the ORB detector in OpenCV was a compromise made because of the limited time and scope of this project. Perhaps it would have been more interesting to compare it against a more state of the art classical detector, or another neural network method of predicting keypoints. It was still valuable to see how the neural network method implemented in this thesis compared to ORB that is popular to use in practice.

The same could be said about the comparison between RANSAC and the consensus maximization network, perhaps a comparison other neural network techniques would also be desirable. But again the RANSAC algorithm in OpenCV is very popular in practice.

7.3 Future work

A very interesting path for future work would be to chain all networks together and train them at the same time. The joint training could possibly lead to better performance overall. Maybe the individual networks would have to be trained separately using the methods described in this thesis before chaining them together and then continue training using a joint loss function. The following is a suggestion for how the networks and their loss functions could be coupled together, illustrated in Figure 7.1.

Two nearby images in the sequence dataset are fed into both keypoint net and depth net. The 2D positions from the keypoint net can be lifted to 3D by sampling the depth maps. The 3D points are fed into the consensus network configured to predict a rigid 3D transformation. The photometric error loss function is implemented as before (Figure 5.2). But instead of using a pose predicting network, the output transform from the consensus network is used instead.

Instead of calculating the point pair and correspondence matrices $\mathbf{G}$ and $\mathbf{C}$ using a known homography, we instead use the rigid 3D transformation to move the 3D points from $\mathbf{I}_s$ and reproject them onto $\mathbf{I}_t$. This uses the same method we use to reproject the image using the rigid transform and depth map, but this time it is done for sparse 3D points. The correspondances are calculated just as in section 5.2.1 but this time using the keypoints from $\mathbf{I}_s$ and the reprojected points from $\mathbf{I}_t$. The keypoint loss function would also be implemented as described in this thesis using the correspondences. Additionally an additional loss term could be added to encourage the keypoint score $\mathbf{s}$ and the inlier prediction $\mathbf{w}$ to be similar.

Having depth maps from two frames in the sequence, and point pairs that should represent the same points in the scene, we can also add a sparse depth consistency loss.

$$\mathcal{L}_{\text{cons}} = \frac{\|\mathbf{D}_t(\hat{\mathbf{p}}_t) - \mathbf{D}_s(\hat{\mathbf{p}}_s)\|}{\mathbf{D}_t(\hat{\mathbf{p}}_t) + \mathbf{D}_s(\hat{\mathbf{p}}_s)} \tag{7.1}$$

This sparse loss reduces scale drifting of the depth maps across frames.

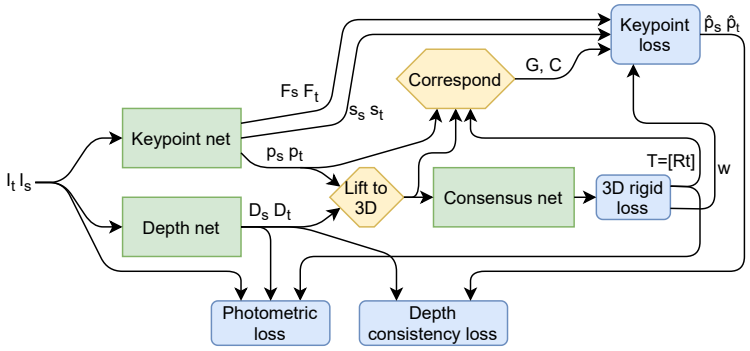

Figure 7.1: *A diagram with a suggestion of how the networks presented in this thesis could be chained together and trained using a joint loss function.*

7.4 Conclusions

The goal of this thesis has been to evaluate a few different unsupervised learning methods in the context of structure from motion. The techniques were combined in new ways and trained on new datasets not used in previous work.

How well do the unsupervised methods from previous research work on new datasets not tested in the original papers? As we have seen, the depth prediction networks can learn to predict depth on the Lyft dataset, not used in previous work. The keypoint prediction network can be trained on images from the Kitti dataset. With some alterations to the consensus maximization network, to improve convergence, it can be trained on the output of the keypoint prediction network. This gives us more confidence that the 3 systems could be chained together and trained jointly in future work. The performance of each technique is detailed in the results chapter.

What are the performance gains of combining different methods from recent research in monocular depth and ego motion prediction? Table 6.2 lists all experiments conducted to compare different techniques in monocular depth and ego motion prediction. The biggest performance gains comes from using SSIM in the photometric error, and using the min() function to combine the per pixel loss across frames. Using a stationary pixel mask seems to be more effective compared to using a predicted explainability mask. Using the edge aware depth smoothess loss term does not improve the metrics much, but does give sharper depth maps upon visual inspection. The techniques with the least impact are depth map normalization and depth map upscaling.

We conclude that the goal of the project has been reached, and that there is a big potential in further research into this field.

Bibliography

[1] Vassileios Balntas, Karel Lenc, Andrea Vedaldi, and Krystian Mikolajczyk. Hpatches: A benchmark and evaluation of handcrafted and learned local descriptors. In *CVPR*, 2017.

[2] P. Christiansen, M. Kragh, Y. Brodskiy, and H. Karstoft. Unsuperpoint: End-to-end unsupervised interest point detector and descriptor. *ArXiv*, abs/1907.04011, 2019.

[3] Daniel DeTone, Tomasz Malisiewicz, and Andrew Rabinovich. Superpoint: Self-supervised interest point detection and description. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR) Workshops*, June 2018.

[4] Martin A. Fischler and Robert C. Bolles. Random sample consensus: A paradigm for model fitting with applications to image analysis and automated cartography. *Commun. ACM*, 24(6):381–395, June 1981. ISSN 0001-0782. doi: 10.1145/358669.358692.

[5] Andreas Geiger, P Lenz, Christoph Stiller, and Raquel Urtasun. Vision meets robotics: the kitti dataset. *The International Journal of Robotics Research*, 32:1231–1237, 09 2013. doi: 10.1177/0278364913491297.

[6] Clement Godard, Oisin Mac Aodha, and Gabriel J. Brostow. Unsupervised monocular depth estimation with left-right consistency. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, July 2017.

[7] Clement Godard, Oisin Mac Aodha, Michael Firman, and Gabriel J. Brostow. Digging into self-supervised monocular depth estimation. In *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*, October 2019.

[8] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*, pages 164–165. MIT Press, 2016. `http://www.deeplearningbook.org`.

[9] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*, page 273. MIT Press, 2016. `http://www.deeplearningbook.org`.

[10] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*, pages 326–327. MIT Press, 2016. http://www.deeplearningbook.org.

[11] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*, pages 80–82. MIT Press, 2016. http://www.deeplearningbook.org.

[12] Vitor Guizilini, Rares Ambrus, Sudeep Pillai, Allan Raventos, and Adrien Gaidon. 3d packing for self-supervised monocular depth estimation. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2020.

[13] Kaiming He, X. Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 770–778, 2016.

[14] Max Jaderberg, Karen Simonyan, Andrew Zisserman, and Koray Kavukcuoglu. Spatial transformer networks. *Advances in Neural Information Processing Systems 28 (NIPS 2015)*, 06 2015.

[15] R. Kesten, M. Usman, J. Houston, T. Pandya, K. Nadhamuni, A. Ferreira, M. Yuan, B. Low, A. Jain, P. Ondruska, S. Omari, S. Shah, A. Kulkarni, A. Kazakova, C. Tao, L. Platinsky, W. Jiang, and V. Shet. Lyft level 5 av dataset 2019. urlhttps://level5.lyft.com/dataset/, 2019.

[16] Tschangho Kim. Automated autonomous vehicles: Prospects and impacts on society. *Journal of Transportation Technologies*, 08:137–150, 01 2018. doi: 10.4236/jtts.2018.83008.

[17] Wilhelm Kirch, editor. *Pearson's Correlation Coefficient*, pages 1090–1091. Springer Netherlands, Dordrecht, 2008. ISBN 978-1-4020-5614-7. doi: 10.1007/978-1-4020-5614-7_2569. URL https://doi.org/10.1007/978-1-4020-5614-7_2569.

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

[19] N.Mayer, E.Ilg, P.Häusser, P.Fischer, D.Cremers, A.Dosovitskiy, and T.Brox. A large dataset to train convolutional networks for disparity, optical flow, and scene flow estimation. In *IEEE International Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016. URL http://lmb.informatik.uni-freiburg.de/Publications/2016/MIFDB16. arXiv:1512.02134.

[20] S. Pillai, Rares Ambrus, and Adrien Gaidon. Superdepth: Self-supervised, super-resolved monocular depth estimation. *2019 International Conference on Robotics and Automation (ICRA)*, pages 9250–9256, 2019.

[21] Thomas Probst, Danda Pani Paudel, Ajad Chhatkuli, and Luc Van Gool. Unsupervised learning of consensus maximization for 3d vision problems. In

The IEEE Conference on Computer Vision and Pattern Recognition (CVPR), June 2019.

[22] C. R. Qi, H. Su, Kaichun Mo, and L. Guibas. Pointnet: Deep learning on point sets for 3d classification and segmentation. *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 77–85, 2017.

[23] Jiexiong Tang, Rares Ambrus, V. Guizilini, S. Pillai, Hanme Kim, and Adrien Gaidon. Self-supervised 3d keypoint learning for ego-motion estimation. *ArXiv*, abs/1912.03426, 2019.

[24] Yan Wang, Wei-Lun Chao, Divyansh Garg, Bharath Hariharan, M. Campbell, and Kilian Q. Weinberger. Pseudo-lidar from visual depth estimation: Bridging the gap in 3d object detection for autonomous driving. *2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 8437–8445, 2019.

[25] Tinghui Zhou, M. Brown, Noah Snavely, and D. Lowe. Unsupervised learning of depth and ego-motion from video. *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 6612–6619, 2017.

[26] Zhou Wang, A. C. Bovik, H. R. Sheikh, and E. P. Simoncelli. Image quality assessment: from error visibility to structural similarity. *IEEE Transactions on Image Processing*, 13(4):600–612, 2004.