Master Thesis

Czech Technical University in Prague

F3

Faculty of Electrical Engineering
Department of Cybernetics

Single View Depth Completion of Sparse 3D Reconstructions

Rakshith Madhavan

Supervisor: doc. Ing. Tomáš Pajdla, Ph.D.
May 2022

MASTER‘S THESIS ASSIGNMENT

I. Personal and study details

Student's name:	**Madhavan Rakshith**	Personal ID number:	**497835**
Faculty / Institute:	**Faculty of Electrical Engineering**		
Department / Institute:	**Department of Cybernetics**		
Study program:	**Cybernetics and Robotics**		
Branch of study:	**Cybernetics and Robotics**		

II. Master’s thesis details

Master’s thesis title in English:

Single View Depth Completion of Sparse 3D Reconstructions

Master’s thesis title in Czech:

Hloubkový obraz z jednoho pohledu a řídké 3D rekonstrukce

Guidelines:

1. Study the approach to single view depth completion [1], 3D reconstruction from multiple images related to visual odometry [2].
2. Suggest an approach to generating dense depth from single images and sparse 3D reconstructions [3, 4].
3. Implement the approach and demonstrate on real data.

Bibliography / sources:

[1] ChaoQiang Zhao, QiYu Sun, ChongZhen Zhang, Yang Tang, Feng Qian Monocular depth estimation based on deep learning: An overview, 2020, Science China Technological Sciences.
[2] Raul Mur-Artal, Juan D. Tardos ORB-SLAM2: An Open-Source SLAM System for Monocular, Stereo and RGB-D Cameras, 2016, IEEE Transactions on Robotics.
[3] Alex Wong, Stefano Soatto Unsupervised Depth Completion with Calibrated Backprojection Layers, ICCV 2021.
[4] Alex Wong, Xiaohan Fei, Stephanie Tsuei, Stefano Soatto, Unsupervised Depth Completion from Visual Inertial Odometry RA-L January 2020, ICRA 2020.

Name and workplace of master’s thesis supervisor:

doc. Ing. Tomáš Pajdla, Ph.D. Applied Algebra and Geometry, CIIRC

Name and workplace of second master’s thesis supervisor or consultant:

Date of master’s thesis assignment: **15.02.2022** Deadline for master's thesis submission: **20.05.2022**

Assignment valid until: **30.09.2023**

doc. Ing. Tomáš Pajdla, Ph.D.	prof. Ing. Tomáš Svoboda, Ph.D.	prof. Mgr. Petr Páta, Ph.D.
Supervisor’s signature	Head of department’s signature	Dean’s signature

III. Assignment receipt

.

Date of assignment receipt	Student’s signature

Acknowledgements

I would like to thank my supervisor doc.Ing. Tomáš Pajdla PhD., for pointing towards the topic of depth completion, and for guiding me in familiarizing myself with the topic.

I would also like to thank my family who have supported me to get me to where I am today. I am grateful for my time in Czech Technical University in Prague, Faculty of Electrical Engineering, and all my endlessly patient lecturers who went out of their way in ensuring that any of my doubts were cleared.

This research was supported by EU H2020 No. 871245 SPRING Project.

Declaration

I hereby declare that I wrote the presented thesis:'Single View Depth Completion of Sparse 3D Reconstructions' on my own and that I cited all the used information sources in compliance with the Methodical instructions about the ethical principles for writing an academic thesis.

In Prague, 20 May 2022.

Abstract

This work outlines a methodology to infer dense depth of a scene from an RGB image, and it's corresponding sparse point cloud using an unsupervised training paradigm and combining it with a visual odometry algorithm such as ORB SLAM [2] in an offline step, to densify the sparse point clouds from its sparse mapping. The network consists of a sparse to dense module, and an encoder to create a 3D positional encoding of the image with a Calibrated Backprojection layer, and the decoder produces the dense depth map. This network is trained without supervision on the data from SLAM by minimizing the photometric reprojection error between frames. Inference is then run on the SLAM Keyframes and sparse depth from its corresponding keypoints to produce dense depth. With thed depth estimate, points from these Key-frames are then back-projected to the point cloud, thus resulting in a denser representation of the scene, especially in low-textured areas where the reconstruction from SLAM ususally fails.

Keywords: densification, sparse, depth-completion, SLAM

Supervisor: doc. Ing. Tomáš Pajdla, Ph.D.
Office B-638 CIIRC Building B,
Jugoslavskych partyzanu 3,
16000 Prague, Czechia

Abstrakt

Klíčová slova:

Contents

Figures

Tables

Terminology and Notation

- Point Cloud - Representation of a space as a collection of 3D points.
- SFM - Structure From Motion
- SLAM - Simultaneous Localization and Mapping
- Visual Odometry(VO) - Process/Algorithm to determine the pose of a moving camera using images
- Keypoints - 2D points in images which are distinct from their neighbours, and are usually the points tracked by SFM/SLAM algorithms like [2]
- Keyframes - Images/frames which are specially selected from the set of all images, which contain some properties such as good covisibility with other keyframes, have sufficient number of keypoints, etc.
- RGBD - Red Green Blue (RGB) pixel Image with Depth values, usually corresponding to each pixel of the image.
- Textured area - Area in an image which is not completely homogeneous in pixel intensities.
- Textureless are - Area in an image which is homogeneous in pixel intensities.
- D - Ground Truth Depth data
- $\hat{D}$ - Estimated Depth data
- d_i or d^i ground truth depth data at pixel i
- $\hat{d}_i$ or $\hat{d}^i$ estimated depth data at pixel i
- $\mathbb{R}^n$ - set of real numbers in n dimensional space
- $\mathbb{R}^n_+$ - set of non-negative real numbers in n dimensional space
- The shape $r \times c$ refers to r: rows, and c: columns when dealing with matrices unless specified otherwise.

- The shape $w \times h$ referes to w: width(columns), and h: height(rows), when dealing with images unless specified otherwise
- N- total number of pixels in an image $N = wh$, unless specified otherwise.
- ORB SLAM: Refers to the works by Rau et al. [3] for monocular systems, and [2] for stereo, rgbd systems

Notes:

- The terms SFM , Visual Odometry, and SLAM are used interchangably in this work, and mean the same thing in the context of this work, although they vary in more general contexts.

Chapter 1
Introduction

Monocular, or stereo cameras are widely used in robotics applications due to their convenient form factors, and the increasing availability of compute capability. Most Visual SLAM or Structure from motion pipelines using mono or stereo cameras are broadly based on tracking a set of keypoints across images, and estimating the camera positions from epipolar geometry, or homographies, triangulating the points, and applying optimization and filtering techniques. Though versions of this pipeline has been adopted with various modifications [55] to achieve real time with high tracking accuracy, even on embedded hardware, an obvious issue is that the resulting 3D scene representation is (very) sparse, usually with a 3D points only for some key-points; while this is sufficient for tasks such as localization, where in the minimal case the camera pose can be estimated from 3 2D-3D correspondences [71] , it is an imprecise representation of the environment, and causes significant issues for tasks such as navigation. When there are textureless obstacles in the path of the robot, they are not modelled accurately for the robot to avoid them during navigation, which results in undesirable collisions. Navigation for ground robots, typically does not use a point cloud representation, but 2D occupancy grids. Techniques of conversion from 3D point clouds to 2D occupancy grids, or occupancy grid creation are not discussed in this work.

This work aims to address the problem using a deep neural network to infer dense depth from an image and a sparse point cloud using calibrated backprojected layers from [1], and coupling this with a VO algorithm, such as ORB SLAM [2] in an offline step to densify the point clouds, particularly in areas of images with low texture. The network is trained on the data obtained from the SLAM algorithm; while overfitting is usually avoided, in this case, the network is made to learn parameters to overfit on the training data, since the inference is on the same data as well.

1.1 Contributions

The main contributions of this work are as follows:

- An overview on historical and state of the art approaches on depth completion detailing the working of several important works.
- A method for the densification of sparse point cloud data from a classical SLAM algorithm [2] using a depth completion network [1] with an overfitting training paradigm, and back-projection to the original point-cloud.
- A depth completion dataset with an RGBD camera on the ARI robot with ground truth for evaluation.

1.2 Structure

The structure of this work following this chapter is as follows:

- Chapter 2 outlines the Overview and Related work
- Chapter 3 describes the structure of the network, and the details of the loss function
- Chapter 4 provides a description of the working of the ORB SLAM algorithm
- Chapter 5 describes the modifications to couple the network with ORB SLAM.
- Chapter 6 describes the setup, and details of the data collection
- Chapter 7 describes implementation details, experiments and their results.
- Chapter 8 concludes this work with insights, and possible directions for future work

Chapter 2
Overview and Related Work

Depth estimation for a scene representation can be broadly divided into three methods:

Geometric Methods: Although geometry-based methods can efficiently calculate the depth values of sparse points, they usually require multiple (>=2) views to compute depth. Recovering 3D structures from multiple views based on geometric constraints has been widely investigated ([51], [55]), and used. Most Structure from Motion(SFM) pipelines and Visual Simultaneous Localization and Mapping (SLAM) algorithms leverage these constraints to estimate the camera positions, as well as the scene representation together simultaneously [56] [57]. In the case of Monocular cameras, we recover the scene 'depth' only upto scale. In the case of stereo cameras, we pre-calibrate the transformation between the two cameras (in some units), and hence we can obtain the scene representation in the same world units as well. The general pipeline for SFM is shown in figure 2.1.
A more detailed overview of these techniques is provided in chapter 4.

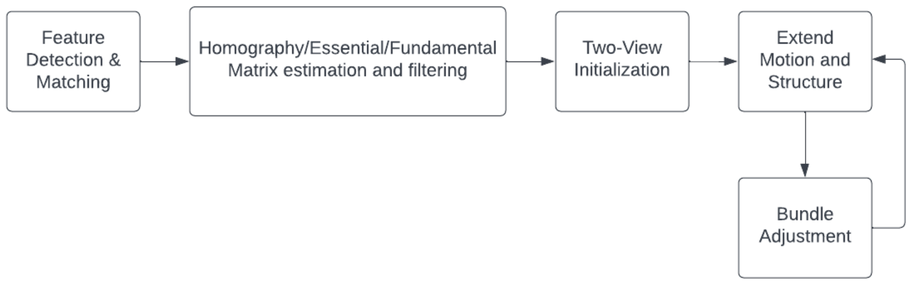

Feature Detection & Matching
Homography/Essential/Fundamental Matrix estimation and filtering
Two-View Initialization
Extend Motion and Structure
Bundle Adjustment

Figure 2.1: General SFM Pipeline

Sensor-based Methods: Depth sensors such as RGB-D cameras, LIDARs, Time-of-Flight(ToF) sensors, etc.. are increasingly used to measure the depth information of a scene directly. However, these sensors come with their own issues such as:Prohibitive cost, Limited functionality such as RGB-D cameras or IR-based ToF sensors working only in the absence of sunlight , prohibitive form factor, and so on..

Monocular Depth Estimation methods: The monocular depth estimation problem seeks to automatically infer a dense depth image from a single color input image, and ,sometimes, it's corresponding sparse depths. This task seems ill-posed without additional view(s) images to enable triangulation. Some classical methods for depth estimation from monocular images include shape-from-shading [77], shape-from-defocus [78], depth inpainting to fill in holes in depth channels of RGBD images using smoothness priors, anisotropic diffusion,etc.. [79] [80] .
With the rapid development and success of deep learning for various computer vision applications such as object detection, semantic segmentation, etc., neural networks have also been used to estimate depth from a monocular image, as well as depth completion of sparse depth. Various neural networks such as CNNs, RNNs, variational auto-encoders, and GANs have been used to address this problem with varying degrees of success.

The deep learning methods can be broadly divided into:

- Supervised
- Unsupervised or semi-supervised
- Based on other methods such as conditional random fields, Adversial learning ,etc.

Of these, this chapter will focus on the supervised, and unsupervised methods, since these have had the most success in various benchmarks.

2.1 Datasets, and common evaluation metrics

[5] provides an overview of the datasets, and commonly used evaluation metrics in monocular depth estimation, and depth completion.

2.1.1 Datasets

- Kitti Depth Completion evaluation [47]: The KITTI dataset is the largest and most commonly used dataset for various sub-tasks in computer vision like optical flow , visual odometry, object detection, semantic segmentation and tracking, etc.. It contains over 93 thousand depth maps with corresponding raw LiDaR scans and RGB images, aligned with the "raw data" of the KITTI dataset. Given the large amount of training data, it allows a training of complex deep learning models for the tasks of depth completion and single image depth prediction. It also provides manually selected images with unpublished depth maps to serve as a benchmark for those two challenging tasks.The Kitti dataset, however, is captured on outdoor data, and performance of methods generally vary between outdoors, and indoors.
- NYU Depth V2 [48]: The NYU-Depth V2 data set is comprised of video sequences from a variety of indoor scenes as recorded by both the RGB and Depth cameras from the Microsoft Kinect. It features 1449 densely labeled pairs of aligned RGB and depth images, 464 new scenes, and 407,024 new unlabeled frames. Each object is labeled with a class and an instance number (cup1, cup2, cup3, etc). The dataset has several components:
    - Labeled: A subset of the video data accompanied by dense multi-class labels. This data has also been preprocessed to fill in missing depth labels.
    - Raw: The raw rgb, depth and accelerometer data as provided by the Kinect.

  The resolution of the RGB images in sequences is 640×480, and they are also down-sampled by half during experiments. Because of the different variable frame rates between RGB camera and depth camera, it is not a one-to-one correspondence between depth maps and RGB images. In order to align the depth maps and the RGB images, each depth map is associated with the closest RGB image at first. Then, with the geometrical relationship provided by the dataset, the camera projections are used to align depth and RGB pairs. Since the projection is discrete, not all pixels have a corresponding depth value, and thus the pixels with no depth value have to be masked off during the experiments.
- VOID Dataset [4]: The dataset was collected using the Intel RealSense D435i camera, which was configured to produce synchronized accelerometer and gyroscope measurements at 400 Hz, along with synchronized

VGA-size (640 x 480) RGB and depth streams at 30 Hz. The depth frames are acquired using active stereo and is aligned to the RGB frame using the sensor factory calibration. All the measurements are timestamped.The dataset contains 56 sequences in total, both indoor and outdoor with challenging motion. Typical scenes include classrooms, offices, stairwells, laboratories, and gardens. Of the 56 sequences, 48 sequences (approximately 47K frames) are designated for training and 8 sequences for testing, from which we sampled 800 frames to construct the testing set. Each sequence constains sparse depth maps at three density levels, 1500, 500 and 150 points, corresponding to 0.5%, 0.15% and 0.05% of VGA size.

- Other datasets such as Cityscapes [49], Make3D, Matterport3D [50] have been used in some works, but are not as popular as the above.

Out of these, the VOID, and NYUV2 datasets are the most relevant for indoor scenes, and have been used for training and testing the model(s) used in this work. The dataset built from the robot mimics the structure of the VOID dataset, and is expanded upon in chapter 6.

2.1.2 Evaluation Metrics

Some of the commonly used evaluation metrics for the depth completion, or estimation task are:

- Root Mean Squared Error: $RMSE = \sqrt{\frac{1}{|N|} \sum_{i \in N} |d_i - \hat{d}_i|^2]}$
- Inverse RMSE: $iRMSE = \sqrt{\frac{1}{|N|} \sum_{i \in N} |1/d_i - 1/\hat{d}_i|^2}$
- RMSE log: $\sqrt{\frac{1}{|N|} \sum_{i \in N} |log(d_i) - log(\hat{d}_i)|^2}$
- Abs Rel: $\frac{1}{|N|} \sum_{i \in N} \frac{|d_i - \hat{d}_i|}{d_i}$
- Accuracies: % of $\hat{d}_\text{i}$ S.T.

$$\max(\frac{\hat{d}_i}{d_i}, \frac{d_i}{\hat{d}_i}) = \delta < threshold$$

- Mean Absolute Error: $MAE = \frac{1}{|N|} \sum_{i \in N} |d_i - \hat{d}_i|$
- Inverse MAE: $iMAE = \frac{1}{|N|} \sum_{i \in N} |1/d_i - 1/\hat{d}_i|$

where N is the total number of pixels in image, d_i is the ground truth depth at pixel i, and $\hat{d}_\text{i}$ is the estimated depth at pixel i.

2.2 Supervised Methods

Supervised learning for the monocular depth estimation task implies learning the weights for the mapping from an RGB image + Sparse Depth to Dense depth using information of labelled input-output pairs. In other words, the objective (loss) function to be minimized consists of comparison between the estimated depths $\hat{D}$ which has a value $\hat{d}_i$ at pixel i and ground truth depths D which has values d_i. This usually contains a form of $\mathrm{L_p}$ loss, which is defined as:

$$L_p = (\sum_{i=1}^{N} |(d_\mathrm{i} - \hat{d}_\mathrm{i})|^p)^{1/p}$$

, or

$$L_p = \frac{1}{N}(\sum_{i=1}^{N} |(d_\mathrm{i} - \hat{d}_\mathrm{i})|^p)^{1/p} \tag{2.1}$$

where N is the total number of pixels in the image (width*height). Often, even the ground truth depth is not fully dense, i.e. it is not available for every pixel; in these cases, N will be the number of pixels for which we have the ground truth depth.

Architectures. One of the earlier works tackling this problem with supervised learning is from Eigen et al. [6], which used two deep network stacks: one that makes a coarse global prediction based on the entire image, and another that refines this prediction locally, to predict the depth map in an end-to-end manner from a single color image with the training loss:

$$L = \frac{1}{N}\sum_{i}(log(\hat{d}_i) - log(d))^2 - \frac{\lambda}{N^2}(\sum_{i}^{N} log(\hat{d}_i) - log(d_i))^2 \tag{2.2}$$

where $\lambda \in [0, 1]$ referes to a balance factor, experimentally set to 0.5.
Laina et al. [8] proposed a fully convolutional architecture based on ResNet [58] trained end to end using a loss based on the reverse Huber function [64]. Roy et al. [9] propsed an architecture combining random forests [74] and CNNs to handle gradually decreasing training sets.
Park et al. [10] proposed estimating non-local neighbors and their affinities of each pixel, as well as an initial depth map with pixel-wise confidences which is then iteratively refined by its confidence and non-local spatial propagation procedure based on the predicted non-local neighbors and corresponding affinities. [11] proposed a lightweight multi-scale guided hourglass network architecture; [12] achieving the state of the art in the KITTI benchmark (as of Dec 2021) proposed a repetitive design in their image guided network to gradually and sufficiently recover depth values. Specifically, the repetition is embodied in both the image guidance branch and depth generation branch. The former branch consists of a repetitive hourglass network, and the latter a repetitive guidance module based on dynamic convolution [19].
[13] proposed a framework to extract the global and local information from

sparse LIDAR point clouds to produce dense depth maps, with an encoder-decoder network based on ERFNet [61] for the former, and two stacked hourglass modules inspired by RESNET [58] for the latter. [14] introduces a network, Sparse Auxillary Network, for both depth completion, or estimation, depending on whether only an RGB image, or RGB image+ sparse depth is available by decoupling the image, and depth map encoding stages. [15] replaces the final 1×1 convolution layer employed in most depth completion networks with a least squares fitting module which computes weights by fitting the implicit depth bases to the given sparse measurements. [16] proposes a network which learns to extract joint 2D and 3D features, which consists of two domain-specific sub-networks that apply 2D convolution on image pixels and continuous convolution on 3D points, with their output features fused in image space. [17] proposes a graph propogation to capture observed spatial contexts. [18] proposes a two-branch network that consists of a color-dominant branch and a depth-dominant branch to exploit and fuse the two modalities. One branch takes as input a color image and a sparse depth map to predict a dense depth map. The other branch takes as inputs the sparse depth, and the estimated dense depth from the previous branch, and outputs a dense depth map; these two depth maps are adaptively fused.

Depth representations. Works such as [26][25] [20] use different depth representations in their works. [26] proposes to deal with depth-smearing across occlusion boundaries with a multi-hypothesis depth representation that explicitly models both foreground, and background depths in the occlusion-boundary regions; [25] proposes an approach to initially densify sparse depth information by figuring out which plane a pixel should lie among a number of discretized depth planes, and to then calculate the final depth value by predicting the distance from the specified plane; [21] proposes a dense mapping framework for sparse Visual SLAM which leverages a compact scene representation proposed in [20]; a dense representation of the scene geometry conditioned on the intensity data from a single image, and generated from a code consisting of a number of parameters.

Indirect Depth completion/estimation. Rather than estimating, or completing the depth directly, some recent works ([27] [28] [29]) have tried to obtain depth estimates indirectly from estimating geometric properties related to the depth, such as surface normals. Zhang et al. [27] uses a fully convolutional neural network built on the back-bone of VGG-16 [59] with symmetry encoder and decoder to predict dense surface normals and occlusion boundaries. These are then combined, and run through global optimization to solve for the depth. The optimization objective to minimize is of the form

$$E = \lambda_D E_D + \lambda_S E_S + \lambda_N E_N B \tag{2.3}$$

where

$$E_D = \sum_{i \in M_{obs}} ||d_i - \hat{d}_i||^2$$

$$E_N = \sum_{i,j \in Neigh} || < v(i,j), Normal(i) > ||^2$$

$$E_S = \sum_{i,j \in Neigh} ||d_i - d_j||^2$$

where E_D measures the distance between estimated depth $\hat{D}(i)$ and observed raw depth $D(i)$ in observations M_{obs}, E_N measures consistency between estimated depth and the predicted surface normal, E_S encourages adjacent pixels to have same depths. $B \in [0,1]$ downweights normal terms based on the predicted probability a pixel is in an occlusion boundary. This non-linear objective is linearized by foregoing vector normalization for tangent vector $v(i,j)$ required for the dot product with surface normal at pixel i, and solved analytically with sparse Cholesky factorization.

X.Qi et al. propose GeoNet [28] to jointly predict depth, and surface normal maps from a single RGB image; by building on top of two-stream CNNs, it incorporates the geometric relation between depth and surface normal via depth-to-normal and normal-to-depth networks. The inference of surface normal from the depth map is formulated as a least squares problem as follows:
For a pixel i, with depth z_i, the 3D coordinates (x_i, y_i, z_i) are computed from its 2D coordinates (u_i, v_i) with the Pinhole camera model. With the assumption that pixels within a local neighborhood of pixel i lie on the same tangent plane, the surface normal estimate $n = [n_x, n_y, n_z]$ should satisfy the over-determined linear system:

$$An = b, \quad subject\ to\ \ ||n||_2^2 = 1 \tag{2.4}$$

, where $A = \begin{bmatrix} x1 & y1 & z1 \\ x2 & y2 & z2 \\ . & . & . \\ . & . & . \\ x_K & y_K & z_K \end{bmatrix} \in \mathbb{R}^{Kx3}$, and b $\in \mathbb{R}^{Kx1}$ is a constant vector; the constrained least squares problem in equation 2.6 has a closed form solution of

$$n = \frac{(A^T A)^{-1} A^T \vec{1}}{||(A^T A)^{-1} A^T \vec{1}||_2} \tag{2.5}$$

where $\vec{1} \in \mathbb{R}^k$. The above is regarded as a fixed weight neural network which predicts surface normal given the depth map.
For the normal-to-depth network; for any pixel i, given its surface normal n_i, and an initial depth estimate z_i. the goal is to refine the depth. Given the 3D point (x_i, y_i, z_i) and its surface normal, the tangent plane P_i can be uniquely determined, which satisfies

$$n_{ix}(x - x_i) + n_{iy}(y - y_i) + n_{iz}(z - z_i) = 0 \tag{2.6}$$

For any pixel j in the neighborhood of a pixel i, with depth z_j assumed to be accurate, the depth estimate of pixel i , z'_{ji} can be computed as:

$$z'_{ji} = \frac{n_{ix}x_j + n_{iy}y_j + n_{iz}z_j}{(u_i - c_x)n_{ix}/f_x + (v_i - c_y)n_{iy}/f_y + n_{iz}} \tag{2.7}$$

Kernel regression is then used to aggregate estimation from all pixels in the neighborhood. [29] adds edge-aware refinement to this to exploit boundary information. [30] trains an encoder-decoder network to predict surface normals, course depth and a confidence measure of inputs which are then inputted to a diffusion refinement module which performs plane-origin distance refinement via anisotropic diffusion based on the assumption that the 3D surface of the scene is constituted by piece-wise planes and the plane-origin distance is piecewise constant. [31] proposed an end-to-end encoder-decode architecture by estimating surface normals as an intermediate representation, for outdoor scenes captured using LiDARs, and uses synthetic Ground Truth genererated from an open urban driving simulator Carla [81] for training.

2.3 Unsupervised & Semi-supervised Methods

Pixel-wise ground truth is expensive to acquire, and most sensors suffer from varying degrees of noise , and inability to capture truly dense depths. Most datasets, as well as techniques described in the above section only compute the loss from those pixels where the ground truth is available; this results in networks at-best learning to reproduce observed depths, but not complete depths that are unobserved, which could and usually have significantly different characteristics.

Garg et al. [7] proposed one of the earlier works on unsupervised depth completion by training on a set of pair of images, source and target, with small, known camera motion between the two, such as a stereo pair. They generate an inverse warp of the target image using the predicted depth and known relative pose, to reconstruct the source image. The photometric error between the reconstructed image and the original source image is then the loss minimized during training: Every training instance $i \in \{1..N\}$ consists of a (stereo) pair of images $\{I_1^i, I_2^i\}$ capture by pre-calibrated stereo rig with focal lengths f, and baseline B.
Predicted depth at pixel $j = \hat{d}_j$, and so the motion of the pixel along the scan line is then $fB/\hat{d}_j$. Using the right image I_2^i, a warp image I_w^i can be generated as

$$I_w^i(j) = I_2^i(j + fB/\hat{d}_j) \tag{2.8}$$

The photometric error is then:

$$E^i_{recons} = \sum_j ||I_w^i(j) - I_1^i(j)||^2 \tag{2.9}$$

They add an L2 regularization $E^i_{smooth} = ||\nabla \hat{d}_j||^2$ on the disparity discontinuities as a prior to deal with the problem of multiple disparities generating

equally good warps, in homogeneous regions of the scene.
The training loss is then:

$$E = \sum_{i=1}^{N} E^i_{recons} + \gamma E^i_{smooth} \tag{2.10}$$

where γ is a parameter determining strength of the regularization forcing the depthmaps to be smooth.
C.Godard et al. [32] improved upon this with a network loosely inspired by the DispNet [60] architecture to generate disparity images by training with an image reconstruction loss, and a consistency loss that enforces consistency between the disparities produced relative to both the left, and right images. A combination of $L1$ and single scale SSIM [62] is used as the photometric image reconstruction cost L_{ap}. This compares the input image I, and its reconstruction I' by the following:

$$L_{ap} = \frac{1}{N} \sum_{i}^{N} \alpha \frac{1 - SSIM(I_i, I'_i)}{2} + (1 - \alpha)||I_i - I'_i|| \tag{2.11}$$

In their work, they use a simplified SSIM with a 3x3 block filter instead of a Gaussian.For the disparity smoothness loss, L_{ds}, they use an L1 penalty on the disparity gradients, and weight this cost with an edge-aware term using the image gradients. The network is trained to predict both left and right disparities, and the left-right disparity consistency loss L_{lr} is a L1 penalty which attempts to make the left-view disparity map be equal to the *projected* right-view disparity map:

$$L_{lr} = \frac{1}{N} \sum_{i} |Disp^l(i) - Disp^r(i + Disp^l(i))| \tag{2.12}$$

At the time, its performance outperformed previous unsupervised methods, as well as some supervised methods on the KITTI benchmark.
Zhou et al. [33] presented an unsupervised learning framework to estimate monocular depth, as well as camera motion from video sequences. Unlike the previous methods discussed above, this was completely unsupervised requiring only a video sequence unlike the others which required some, albeit minimal, supervision in the form of pre-calibrated stero rigs. It uses single-view and multi-view pose networks to regress the pose between the frames, and trains on a loss based on warping nearby views to the target using the estimated depth, and poses.
Let $< I_1, .., I_N >$ denote a training sequence of frames with one of the frames being the target I_t view, and the rest source views I_s for $s \neq t$. The view synthesis objective is formulated as:

$$L_{vs} = \sum_{s} \sum_{i} \hat{E}^i_s |I^i_t - \hat{I_s}^i| \tag{2.13}$$

where i indexes over pixels, $\hat{I_s}$ is the source view I_s warped to the target coordinate frame based on the estimated depth, and relative pose, and $\hat{E}_s$

is a per-pixel soft mask for each target-source pair indicating the belief in where direct view synthesis will be succesfully modeled. They use a L_1 penalty on the second order gradients for smoothness L_{smooth}, and add a regularization term $L_{reg}(\hat{E}_s)$ to encourage nonzero predictions by minimizing the cross-entropy loss with constant label 1 at each pixel location.
Based on the above work, Yang et al. [34] incorporates an edge-aware depth-normal consistency constraint in the network. The mutual conversion between depth and normal is solved by including a depth-normal layer, and a normal-depth layer in the network, and the network achieves a higher accuracy than [33].
F.Ma et al [35] proposed an encoder-decoder network which predicts dense depth directly from RGB image (when available) and sparse depth. It is, similar to the other unsupervised training works, trained on view synthesis and the photometric loss. However, to compute the relative poses between frames, rather than using a pose network, or pre-calibrated rigs they use Pnp [72] with RANSAC [73] to estimate relative transformation between the current frame and the nearby frame; using matched feature correspondences extracted from RGBD (current) and RGB (nearby) respectively. Sivakumar et al. [37] proposed a CNN to upsample a series of sparse depth measurements based on contextual clues from a high resolution image which is trained in a semi-supervised manner with optional stereo supervision. Yang et al. [38] estimates the posterior distribution of dense depth map by using a Conditional Prior Network that associates a probability to each depth value in an image, and combines it with a likelihood term that uses the sparse depth measurements. The training is either supervised, with the training loss as a prediction error:

$$L(w) = \sum_{i=1}^{N} ||\phi(z_i, I_i; w) - d_i||^{\gamma} \tag{2.14}$$

where ϕ is the map from sparse depth z and image I to dense depth. They fix the parameters w and γ to 1 during training. In the case of unsupervised training, they use a Conditional Prior Network (CPN), which infers the probability of an optical flow given a single image, to score the compatibility of a dense depth with the given image based on previous observed data. If additional sensory data is available during training, such as stereo pairs, then disparity supervision is used.
Recently, methods such as [4], [1] use the sparse depth, and camera poses estimated from visual odometry (or VIO) which typically consist of very sparse point clouds using photometric error as the supervisory signal. [4] first constructs a piece-wise planar scaffolding of the scene, which it then uses to infer dense depth. To compute the scaffolding, relying on a prior that surfaces are locally smooth, and piecewise planar , they apply a lifting transform to the sparse depth, and then compute its convex hull [75] of which the lower envelope is taken as the Delaunay triangulation of the points, resulting in a triangular mesh. Each surface is then approximated using linear interpolation within the Barycentric coordinates, and the resulting scaffolding is projected

back into the image plane.A pose network is used with an encoder-decoder network based on VGG-11, or VGG-8 [59] (for the light-weight model) then refines this scaffolding, along with its RGB image input to recover the dense depth, and is trained on the following loss:

$$L = w_{ph} L_{ph} + w_{sz} L_{sz} + w_{pc} L_{pc} + w_{sm} L_{sm} \tag{2.15}$$

where L_{ph} is the photometric consistency which is a combination of average per-pixel reprojection residual with an L1 penalty and SSIM [62], L_{sz} is an L1 norm of estimated depth, and sparse depth, L_{pc} is the pose consistency loss, and L_{sm} local smoothness, an $L1$ penalty of the first order gradients in the x, and y directions, and the ws are the associated weights. Since the other losses have been discussed previously, only the pose consistency loss for the pose network is expanded here: For an ordered pair of images (I_t, I_τ), the relative pose $g_{\tau t} \in SE(3)$ is the relative (forward) pose, and $g_{t\tau} \in SE(3)$ the backward pose, which is the inverse of of the forward pose. The forward-backward pose penalizes the deviation of the composed pose from identity.

$$L_{pc} = ||log(g_{\tau t}.g_{t\tau})||_2^2 \tag{2.16}$$

where log: $SE(3) \rightarrow se(3)$ is a logarithmic map. They also present the VOID dataset, which is introduced in section 2.2.1
[1] uses an encoder-decoder architecture with calibrated backprojected-layers trained on the photometric reprojection error, for unsupervised training on visual-inertial odometry; which is the network used in this work. It is expanded in detail in the following chapters

2.4 Combination with SLAM

Monocular depth estimation based on deep learning has been applied in SLAM to improve mapping, recover absolute scale, etc. [39] improves upon the mapping in SVO [40] by initializing the mean, and variance at feature locations according to depth prediction from a monocular depth estimation network [32]. This reduces the depth uncertainty of an initialized map point leading to reliable feature correspondence between views, and faster convergence to the true depth. This method outperforms the SVO mapping on the KITTI dataset [47]. Yin et al. [41], and Yang et al. [42] used depth estimation to recover the absolute scale of monocular VO.
[20] uses an image-conditioned autoencoder trained on depth images to discover a optimizable 'code', a compact representation, with a small number of parameters which describes the dense depth map at a keyframe. For a keyframe based SLAM, the camera poses with these 'code' representations can be jointly optimized to estimate a dense scene. [22] uses a [20]-like VAE conditioned on a grayscale intensity image, and a sparse depth map, with a system tightly coupled with filtering based VIO and simultaneously estimates scene geometry, and camera trajectory in the same state space.

[21], similar to this work, in that it takes as input the camera poses, keyframes, and sparse points from the ORB-SLAM3 [23], in this case, mapping and predicts a dense depth image for each keyframe. They, however build on [20] and use a variational autoencoder (VAE) trained on intensity, sparse depth, and reprojection error images to predict an uncertainty aware dense depth map in an online process. Consecutive depth maps are refined through multi-view optimization using fixed camera poses provided by the SLAM system. They run two threads in parallel, a SLAM thread running ORB-SLAM3 [23] tracking and mapping (described in more detail in chapter 4), and after every ORB local bundle adjustment transfers data from a window of four keyframes to the second thread. When the second thread, or Dense-Mapping thread, receives the keyframes' data runs the depth completion VAE using the sparse depth and reprojection error images to predict an initial dense depth map, and also generates low dimensional latent codes . They then run multi-view optimization based on factor-graph optimization from [24] optimizing only the codes, and not poses.

Merrill et al. [43] propose a lightweight network based on the hourglass FastDepth architecture [35] to add to their EKF-based OpenVINS [44] VIO algorithm by projecting tracked sparse features to an image similar to this work. Sartipi et al. [45] uses a surface normal network [Deeplidar] with sparse data from VI SLAM.

Table 2.1: List of a some works with open source code bases; classifying whether they were designedfor indoor/outdoor settings, trained supervised or unsupervised, with information on sensor type (eg. LiDARs, RGBD cameras, VIO etc.) for acquiring sparse depth information

Works	Setting	Sensing method	Training Method	Code location
Hu et al. [18]	Outdoor(KITTI)	LiDAR	Supervised	github
Zhang et al. [27]	Indoor	RGB-D	Supervised	github
Park et al. [10]	Indoor/Outdoor	RGBD/LiDAR	Supervised	github
Gansbeke et al. [13]	Outdoor(KITTI)	LiDAR	Supervised	github
Qiu et al. [31]	Outdoor(KITTI)	LiDAR	Supervised	github
Imran et al. [26]	Indoor/Outdoor	RGBD/LiDAR/Synthetic	Supervised	github
Zhu et al. [46]	Indoor	RGBD	Supervised	nvidia(future)
Shivakumar et al. [37]	Outdoor	LiDAR	Supervised	github
Qi et al. [28][29]	Indoor	RGBD	Supervised	github
Ma et al. [36]	Indoor	RGBD/SLAM	Supervised	github
Mat et al. [35]	Outdoor	LiDAR	Unsupervised	github
Zhou et al. [33]	Indoor/Outdoor	Video	Unsupervised	github
Yang et al. [38]	Outdoor	LiDAR	Unsupervised	NA
Wong et al. [4]	Indoor/Outdoor	VIO	Unsupervised	github
Wong et al. [1]	Indoor/Outdoor	VO/SLAM	Unsupervised	github
Czarnowski et al. [24] (SLAM)	Indoor/Outdoor	SLAM	Unsupervised	github

Chapter 3

Network Architecture

This work uses the network from [1] by Wong et al., whose architecture and the loss function used for training are described in this chapter. The network takes in as input an RGB image, its intrinsic calibration matrix (see Appendix A), and the sparse depth image corresponding to the RGB image, and outputs the dense depth image with each pixel containing the depth value corresponding to the pixel in the RGB image. The network consists of a Sparse-to-Dense (S2D) module, a two-branch encoder for RGB 3D Encoder, and Depth Encoder, and a Decoder.

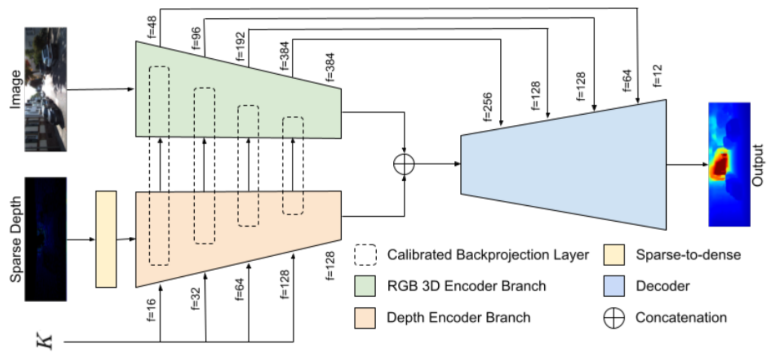


Figure 3.1: Network architecture from [1]. f here is the number of filters or output channels used in the convolution operations

The sparse-to-dense module learns a dense encoding representation of the sparse depth; A Calibrated Backprojection (KB) layer backprojects each pixel in the image to a 3D space. The resulting 3D positional encoding is concatenated with the image descriptor and the previous layer output to yield the input to the next layer of the encoder. The encoder consists of 5 resolutions as shown in figure 3.1 with the number of convolution filters for the image encodings are [48, 96, 192, 384, 384], and for the depth encodings [16, 32, 64, 128, 128]

A decoder uses skip-connections, and upsampling with a series of Up-Convolution, or transposed convolutions (see section B.3) and produces the output dense depth image.

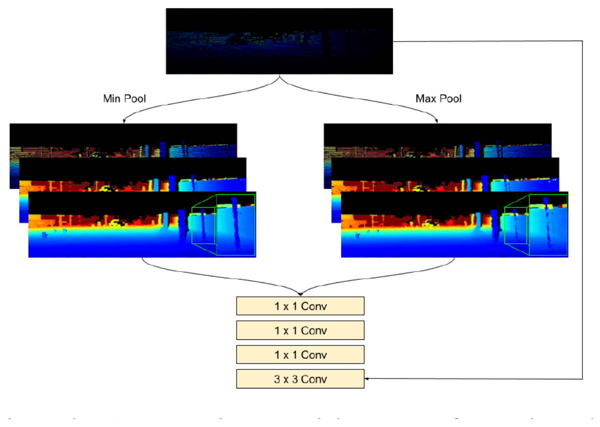


Figure 3.2: Sparse-to-Depth module. figure from [1]

3.1 Sparse-to-Dense Module(S2D)

The S2D consists of various pooling [appendix], and convolutional [appendix] layers to yield a dense representation of the sparse depth image. Specifically, it performs multi-scale densification using a series of min and max pooling layers with different kernel sizes chosen on the basis of point cloud sparsity. Experiments with these kernel sizes are described in chapter 7. In practice, the max pooling(s) are done with kernel of size S with a single stride, and half-padding(for $S/2$ pixels) with $-\infty$, and min pooling is done similarly with a half-padding of ∞ .
The outputs of the pooling layers are concatenated and fed into three 1×1 convolutions , whose result is fused with the original sparse depth z via a 3×3 convolution layer giving a quasi-dense depth representation.
Due to the, by nature, sparsity of the input data, the min pooling layers avoid pooling zeros or invalid depth, by setting all such values to infinity.

$$z'(i) = \begin{cases} z(i) \; if \; z(i) > 0 \\ \infty \; otherwise \end{cases} \quad i \in N \tag{3.1}$$

where N is the total number of pixels, and i is a pixel in the image. z' is then fed to the min-pool. For kernel-sized or larger regions where all values in z' are ∞ due to sparcity, for pooling operation $p(i)$;

$$p_{min}(i) = \begin{cases} p(i) \; if \; p(i) \neq \infty \\ 0 \; otherwise \end{cases} \tag{3.2}$$

The pooling, and convolution operations are described in appendix B.

3.2 KBNet Architecture

When a network is trained without supervision on the photometric reprojection error, described in section 3.3, without prior information about the intrinsic

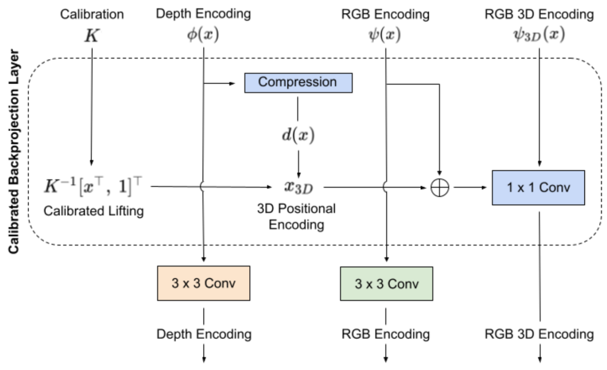


Figure 3.3: Calibrated Backprojected layer; figure from [1]

calibration K, they implicitly learn the calibration parameters of K. This results in the possibility of generalization being extremely low, no matter how good the training data. In order to address this, as well as to produce depth estimates that better respect object boundaries, and reduce the bleed effect observed when a depth map is backprojected to a 3D point cloud, [1] proposed calibrated backprojected layers; practically, this also results in a lighter network with fewer layers and parameters to achieve state of the art results.

Calibrated Backprojected (KB) Layers. take as input the feature maps of image encodings (which are of the form $image\_shape \times f$ where f is the number of filters in the previous layer as can be seen in figure 3.1), depth encodings (of the form $image\_shape \times f$), and a 3×3 calibration matrix K (A.5). This is realized in the following way:
The direction vectors for each coordinate i in the 2D image (u^i, v^i), are computed with the calibration matrix, with

$$\vec{X^i_{dir}} = \lambda K^{-1} \begin{bmatrix} u^i \\ v^i \\ 1 \end{bmatrix} \tag{3.3}$$

The above gives, for each pixel in the image, a direction vector along which the actual 3D point will lie in since any point on the vector will project to the same 2D point (u^i, v^i) in the image. If the depth d^i is known, the coordinates of the point with respect to the camera coordinate frame (frame $C(\vec{c1}, \vec{c2}, \vec{c3})$ in figure A.2) can be obtained with:

$$X^i = d^i K^{-1} \begin{bmatrix} u^i \\ v^i \\ 1 \end{bmatrix} \tag{3.4}$$

The depth encodings for each pixel $\phi^i \in \mathbb{R}^f$ are projected from $\mathbb{R}^f$ to a $\mathbb{R}$ with a trainable 'compression module' which can be seen as:

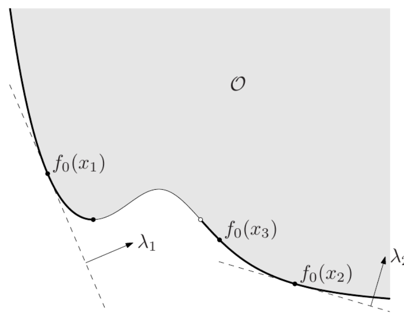
$$d^i = q^T \phi^i \tag{3.5}$$

where q^T is the trainable 'compression module' which is performed with 1×1 convolution operations. This d^i from (3.5) is used in (3.4) to obtain a 3D positional encoding for each pixel X^i_{3D}
Here $i \in H \times W$ corresponds to the resolution of the first image , that decreases by a factor of 2 in each layer until the 5th at $H/32 \times W/32$. Hence, the intrinsic parameters, must also be scaled by the same scale factor according to the resolution. This 3D positional encoding is concatenated with the image encoding, and if available the output of the previous KB layer. This is then fused together by a 1×1 convolution to yield the output RGB 3D encoding. This is fed to the next layer, and also replaces the typical RGB skip connection to the decoder. Finally, the output depth and image encodings of the KB layer are produced by convolving separate 3×3 kernels which are also then passed to the next layer as inputs.

The overall system during training, which in [1] uses a Posenet, or a pose-network for computing relative poses, is shown in figure 3.5

3.3 Loss function and Training

The training is a vector optimization problem where we optimize the vector $\vec{L} = \begin{bmatrix} l_{ph} \\ l_{sz} \\ l_{l_s m} \end{bmatrix}$,where l_{ph} denotes photometric consistency, l_{sz} the sparse depth consistency, and l_{sm} a local smoothness.
Since we compare with respect to the cone $K : \mathbb{R}^3_+$, whose dual cone K^* is also $\mathbb{R}^3_+$, for any weights $\vec{W} \in K^* : \mathbb{R}^3_+$, we can get a Pareto optimal solution [54], but the weights used for this scalarization can be seen as the importance we give to each variable(loss type) in the vector.



Scalarization. The set $\mathcal{O}$ of achievable values for a vector optimization problem with cone $K = \mathbf{R}^2_+$. Three Pareto optimal values $f_0(x_1)$, $f_0(x_2)$, $f_0(x_3)$ are shown. The first two values can be obtained by scalarization: $f_0(x_1)$ minimizes $\lambda_1^T u$ over all $u \in \mathcal{O}$ and $f_0(x_2)$ minimizes $\lambda_2^T u$, where $\lambda_1, \lambda_2 \succ 0$. The value $f_0(x_3)$ is Pareto optimal, but cannot be found by scalarization.

Figure 3.4: Visualization of Pareto optimal for $\mathbb{R}^2$ with respect to the cone $\mathbb{R}^2_+$; figure from [54]

An example for scalarization, and Pareto optima for $\mathbb{R}^2$ vector is shown in figure 3.4

3.3.1 Loss function

This vector objective is scalarized [CO] from $\mathbb{R}^3$ to $\mathbb{R}$ with the weights $\vec{W}$ to reach a Pareto optimal solution formulated as:

$$L = w_{ph}l_{ph} + w_{sz}l_{sz} + w_{sm}l_{sm} \tag{3.6}$$

Photometric Consistency Loss (l_{ph}). discussed briefly in section 2.3 is described more in detail here.

Consider two images I_1, and I_2 with respective intrinsic calibration matrices K_1, K_2, and whose relative pose is represented as $p \in SE(3)$, i.e. p is the pose of camera frame of I_1 in the frame of I_2 or an affine transformation which transforms a point from the camera frame of I_1 to that of I_2.
Given this information, the principle of photometric loss measure is to reconstruct image I_1 from image I_2 as $\hat{I_{12}}$ and compare I_1 with this $\hat{I_{12}}$
Consider a point X with the pixel coordinates (u_1, v_1) in I_1. This point will have some pixel coordinates (u_2, v_2) in I_2. To get this (u_2, v_2): First, we re-project this point to a ray/direction vector in the camera coordinate frame of I_1 with (3.3)

$$\vec{x_1^{dir}} = \lambda K_1^{-1} \begin{bmatrix} u_1 \\ v_1 \\ 1 \end{bmatrix} \tag{3.7}$$

Then, we use the estimated depth value $\hat{d_i}$ for that pixel, to get the 3D point X_1 which projects to it (in the frame of camera I_1) with:

$$X_1 = \hat{d_i} K_1^{-1} \begin{bmatrix} u_1 \\ v_1 \\ 1 \end{bmatrix} \tag{3.8}$$

Now, by constructing a 3×4 projection matrix P_2 with K2, and the relative pose p with (A.11) we can recover the projection of this point in the image I_2 with (A.10):

$$\lambda_2 \begin{bmatrix} u_2 \\ v_2 \\ 1 \end{bmatrix} = P_2 X_1 \tag{3.9}$$

or

$$\lambda_2 \begin{bmatrix} u_2 \\ v_2 \\ 1 \end{bmatrix} = P_2 \hat{d_i} K_1^{-1} \begin{bmatrix} u_1 \\ v_1 \\ 1 \end{bmatrix} \tag{3.10}$$

where

$$P_2 = \begin{bmatrix} K_2 & \vec{0} \end{bmatrix} p \tag{3.11}$$

and the coordinates of this point in the image I_2 is: $[u_2, v_2]^T$.
Thus, for the reconstructed image we get

$$\hat{I_{12}}(u_1, v_1) = I_2(u_2, v_2) \tag{3.12}$$

From this, it is easy to see that the more accurate the depth estimate (and relative pose) is, the more similar $\hat{I_{12}}$ will be to I_1.

For a frame I_t at t, the photometric consistency loss hence penalizes the dissimilarity using a combination of L_1 penalty and $SSIM$ [62]:

$$l_{ph} = \frac{1}{|N|} \sum_{\tau \in T} \sum_{i \in N} w_{co} |\hat{I^i_{t\tau}} - I^i_t| + w_{st}(1 - SSIM(\hat{I^i_{t\tau}}, I^i_t) \tag{3.13}$$

where N is the total number of pixels in the image, $T = \{t-1, t+1\}$, and w_{co},w_{st} are weights denoting color consistency with the L_1 penalty, and structural consistency with the $SSIM$.
In [1], they use a Pose-network to estimate the relative pose ,p, between frames and jointly optimize that along with the depth completion network without additional supervision. In this work however, the pose-network is not used and the relative pose is obtained from Orb-Slam [2] instead. How this is done is described in section 3.3.2, and tests with poses from a pose-network as well as orb-slam are described in chapter 7.

Sparse Depth Consistency. Minimizing the above error will reconstruct the scene structure up to an unknown-scale. To ground the predictions to a metric scale, that of the sparse depth, the sparse depth consistency is enforced as an L_1 penalty for the difference between the depth estimate at a pixel $\hat{d}_i$ and the sparse depth at that pixel z_i, for $i \in N_z$ where N_z is the set of all pixels for which sparse depth exists.

$$l_{sz} = \frac{1}{N_z} \sum_{i \in N_z} |\hat{d}_i - z_i| \tag{3.14}$$

Local Smoothness. The necessity to enforce local smoothness was briefly discussed in section 2.3. This is enforced by minimizing the first order gradients in the x direction : ∂_X, as well as the y direction: ∂_Y. The terms are also weighted using their respective image gradients $\lambda_X = e^{-|\partial_X I^i_t|}$ and $\lambda_Y = e^{-|\partial_Y I^i_t|}$ to allow discontinuities along object boundaries.

$$l_{sm} = \frac{1}{N} \sum_{i \in N} \lambda^i_X |\partial_X \hat{d^i}| + \lambda^i_Y |\partial_Y \hat{d^i}| \tag{3.15}$$

3.3.2 Training

The overall system while training in [1] is shown in figure 3.5

Each training sample contains the following data: image I_t, images I_τ where $\tau \in T : \{t-1, t+1\}$, the sparse depth map z corresponding to I_t, the intrinsic calibration matrix K, and the relative poses between I_t and I_τ obtained from ORB-SLAM2 [2].
[1] does not use poses in the training data, but obtains the relative pose p as described in 3.3.1 from a pose network based on PoseNet. The supervisory loss for the pose network is from (3.13). For two images I_t, I_τ the pose network directly regresses the relative pose p, but most SLAM or VO algorithms,

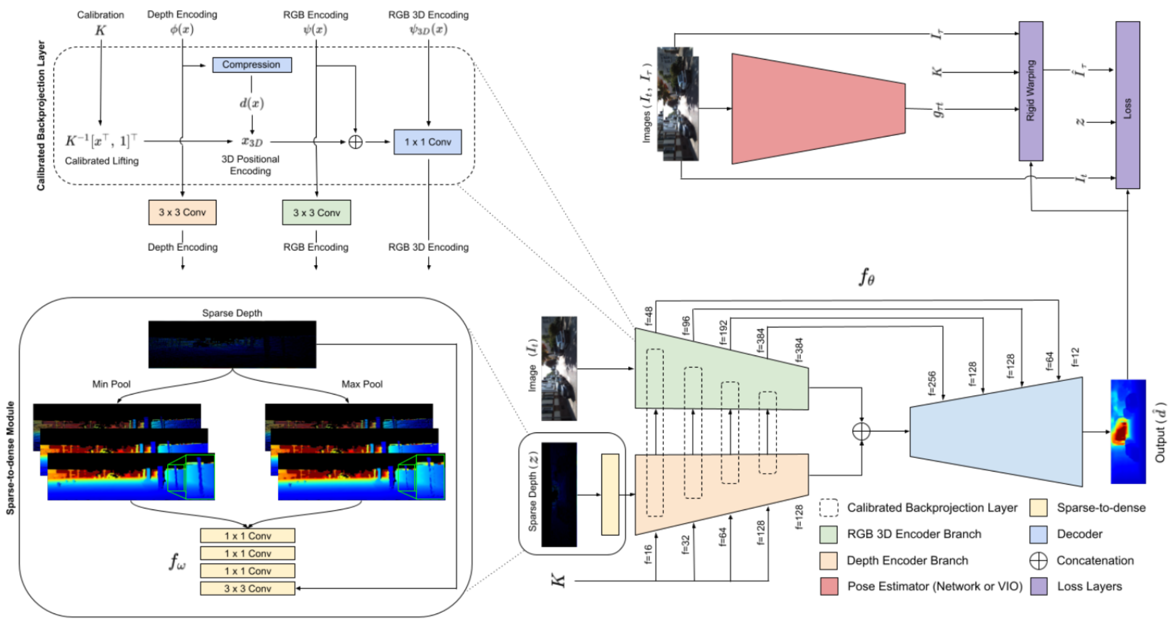


Figure 3.5: The system during training in [1]: Each training sample contains (I_t, $I_{\tau \in T}$, sparse depth (z), intrinsic calibration K). The original work uses a PoseNet to get relative depths for l_{ph} but this work uses poses from ORB-SLAM;

including ORB-SLAM store all the camera poses with respect to some global frame, usually initialized as the coordinate frame of the initial frame. How this is done is detailed in chapter 5 and the implementation details, and experiments with training are discussed in chapter 7.

Chapter 4
ORB SLAM

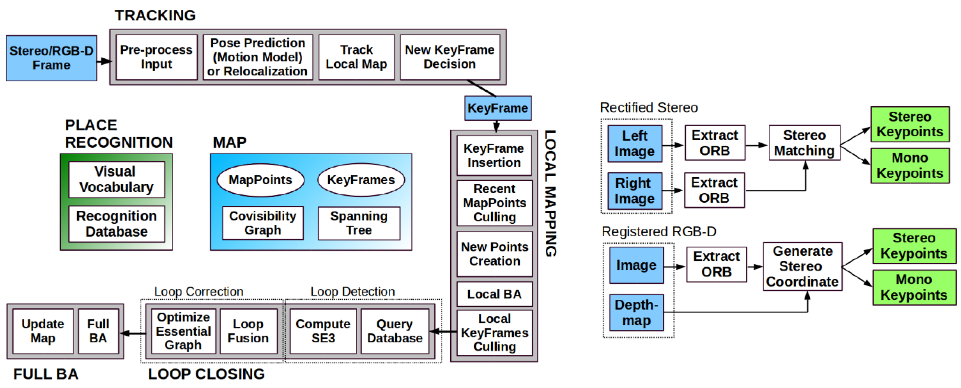
This work uses ORB-SLAM [2] for obtaining the camera poses, as well as the sparse point cloud. ORB-SLAM is one of the most widely used open-source SLAM/VO algorithms due to its robustness, versatility and speed. It uses ORB features [65], based on the FAST keypoint detector [66] and BRIEF descriptors [67], for real time feature extraction and matching, and a Bag of words (BOW) implementation [82] for image retrieval used during localization with an optimization based approach for map generation and tracking. The details of the workings of [2] will be presented here, and the modifications made for the coupling with the network described in chapter 3 will be presented in chapter 5.

4.1 Overview

An overview of the working of ORB SLAM can be seen in figure 4.1. The case of an RGBD input is discussed here since the testing for this work was done with inputs from an RGBD camera. There are three threads running simultaneously, for the *tracking*, *local mapping*, and *loop closing*.

In the tracking. thread for an RGBD input, the ORB features (see 4.3.1) are first extracted incuding information about their 2D location (u, v) and the depth corresponding to that point (d). The points from a single RGBD image are then projected as stereo coordinates (see section 4.3.3). The map is then initialized with the first frame as a 'keyframe', and its ORB 'keypoints' as points of the map. For subsequent frames, the motion is tracked with a constant velocity motion model to predict the camera pose, and the poses are optimized with full Bundle Adjustment (see section 4.6).
If the tracking is lost, a place recognition module using bag-of-words is emplyed to perform a global relocalization. The system maintains a covisibility graph [8 in ORBSLAM2] that links any two keyframes observing commonm points and a minimum spanning tree (see Appendix D.2 connecting all keyframes. Finally, the tracking thread decides if a new keyframe is inserted into

The Local Mapping. thread processes new keyframes and performs local motion-only Bundle Adjustment (section 4.6.1). New correspondences for



(a) System Threads and Modules. (b) Input pre-processing

ORB-SLAM2 is composed of three main parallel threads: tracking, local mapping and loop closing, which can create a fourth thread to perform full BA after a loop closure. The tracking thread pre-processes the stereo or RGB-D input so that the rest of the system operates independently of the input sensor.

Figure 4.1: Components of ORB-SLAM; figure from [2]

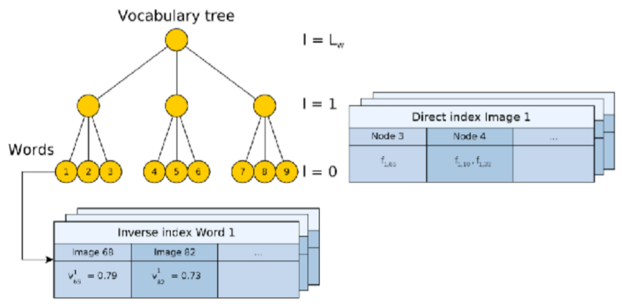
unmatched ORB points in the new keyframe are searched in connected keyframes in the covisibility graph [76]to triangulate new points. Based on information gathered during the tracking, a point culling policy is applied to filter out some lower-quality points, as well as redundant keyframes.

Loop Closing. thread searches for loops (in the trajectory) with every new keyframe. If a loop is detected, the drift accumulated is estimated, and bosed sides of the loop are aligned and duplicated points fused. Lastly, a pose-graph optimization (4.6.3) is performed over the Essential Graph, a sparser subgraph of the covisibility graph.

4.2 Image Retrieval with BoW

Image retrieval is the task of finding the most similar image from a database, to that of a query image. There are various methods tacking this, with deep learning based methods recently outperforming conventional ones in terms of accuracy. However, techniques like the Bag of Words still perform with high accuracy and much faster retrieval times, so a Bag of Words approach is what is used in the ORB SLAM works.
It involves constructing a 'dictionary' or a 'Vocabulary' where the 'words' are some visual features. This vocabulary is obtained from a number of images, by clustering a set of features in an image into a 'word'.
An image is then stored, or represented in the database as a histogram of the occurances of these 'words' from the Vocabulary. During query time, the query image is represented as a histogram of these words as well, and the nearest neighbor histogram is found, using a TF-IDF distance measure.

ORB SLAM uses the DBoW2 [82] implementation with a vocabulary created offline with ORB descriptors (4.3.1) extracted from a large set of images. They also add various implementation tricks to add to the performance described in their paper [3].



Example of vocabulary tree and direct and inverse indexes that compose the image database. The vocabulary words are the leaf nodes of the tree. The inverse index stores the weight of the words in the images in which they appear. The direct index stores the features of the images and their associated nodes at a certain level of the vocabulary tree.

Figure 4.2: BoW Vocabulary tree

4.3 Tracking

4.3.1 ORB Features

ORB [65] are binary features invariant to rotation and scale (in a certain range), resulting in a very fast recognizer with good invariance to viewpoint.

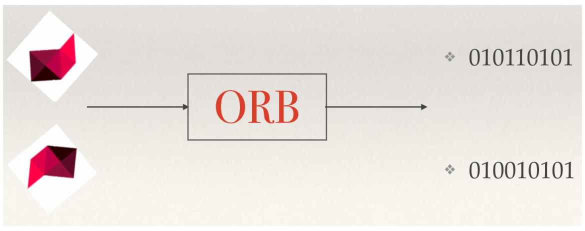


Figure 4.3: An example of how ORB SLAM would work. Takes in as input a image patch, and outputs a binary string which is rotation and scale invariant. In this example, the two image patches are rotated, but the descriptor remains the same

The feature points are extracted with a FAST keypoint detector [66] at 8 scale levels with a scale factor of 1.2 , and the descriptor is based on the BRIEF descriptor [67]; the name ORB is for Oriented FAST and Rotated BRIEF.

4.3.2 Covisibility, and Essential Graphs

Covisibility graph. is an undirected weighted graph where each node is a Keyframe and an edge between Keyframes exist if they share observations of the same map points (at least 15 in the original implementation). The weight θ of the edge is the number of common map points.

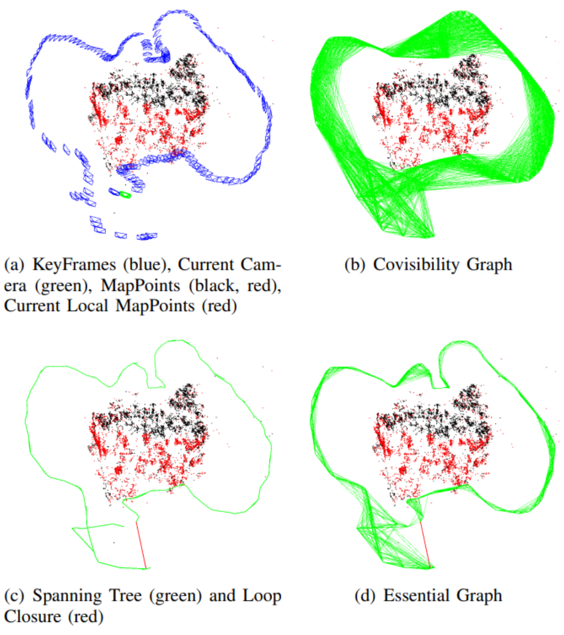


Figure 4.4: Example of some components of ORB-SLAM2; figure from [ORB-SLAM1]

Essential Graph. is a subgraph of a covisibility graph that contains all its nodes, but not the edges. The system incrementally builds a spanning tree (D.2) from the initial keyframe which provides a connected subgraph of the covisibility graph with minimal number of edges. When a new Keyframe is inserted, it is included in the tree linked to the keyframe with which it shares the most observartions, and when a Keyframe is removed by the culling policy, the system updates all the nodes (keyframes) linked to it. The *Essential Graph* contains this spanning tree, the subset of edges from the covisibility graph with high covisibility ($\theta_{min} >= 100$) and loop closure edges. The essential graph is used for the pose graph optimization (4.6.3).

See figure 4.4 for an example of covisibility graph, spanning tree, and associated Essential graph.

4.3.3 Stereo points

A stereo point contains the following information: $x_s = (u_l, v_l, u_r)$ where (u_l, v_l) is the same as (u, v) of the 'left camera' for an imaginary stereo set-up and u_l the horizontal coordinate for the 'right image'. For this imaginary stereo set-up we assume it to be rectified left and right images, and hence $v_r = v_l$ so we do not mention it explicitly. The coordinate u_r is generated with the depth as:

$$u_r = u_l - \frac{f_x b}{d} \tag{4.1}$$

where f_x is the horizontal focal length, and b is some fixed baseline. This is done in the original work so that it is easier to have a single setup for both

stereo, and RGBD cameras.
The mono keypoints contain only (u_l, v_l), for pixels where there is no depth value available.

4.3.4 Pose tracking

After the map has been initialized with the points from the first frame, a constant velocity motion model [83] is used to predict the camera pose and perform a guided search of the map points observed in the previous frame. If insufficient matches were found, it implies that the motion model was violated; a wider search is used for the map points around their position in the last frame. The pose is then optimized with the found correspondences.

4.3.5 Relocalization if tracking is lost

If the tracking of the features across frames is lost for a frame, that frame is converted into a bag of words representation (see 4.2), and the databse consisting of all the previously tracked keyframes is queried for candidates C for global relocalization. ORB 2D features of the query frame, I_q are matched with 2D keypoints which contain a corresponding 3D points in the global map from database image $I_d \in C$.
Then, Pnp[72] with RANSAC [73] is run to find camera poses and their inliers for each of the candidate keyframes. If there is a valid camera pose (with enough inliers) the pose is optimized and a guided search is done for finding more matches with the map points of the candidate frame. Finally the camera pose is again optimized, and if supported with enough inliers, tracking procedure continues.

4.3.6 Local map tracking

A local map contains a set of keyframes K_1 that share map points with the current frame, and a set K_2 with neighbors to the keyframes in K_1 in the covisibility graph (4.3.2). It also has a reference keyframe $K_{ref} \in K_1$ which shares most map points with the current frame. Each map point seen in K_1, and K_2 is searched in the current frame as:

1. The projection x of the map point as in Appendix A is computed, and discarded it if the image coordinates are out of bounds of the image resolution.

2. The angle between the current viewing ray $\vec{v}$ (see (3.3)), and the map point mean viewing direction $\vec{n}$ is computed, and map point discarded if $\vec{v}^T\vec{n} < cos(60°)$

3. Distance d of map point from camera center is calculated as L_2 distance and point is discarded if it is out of the scale invariance region, i.e. $d \notin [d_{min}, d_{max}]$

4. Scale in the frame computed by the ratio d/d_{min}

5. The representative descriptor D of the map point with the still unmatched ORB features in the frame, at the predicted scale, and near x are compared to associate the map point with the best match.

The camera pose is then optimized with all the map points in a full bundle adjustmnent.

4.4 Local Mapping

The local mapping thread handles the following with every new Keyframe K_i :

4.4.1 KeyFrame Insertion

The covisibility graph is updated with a new node for K_i and updating relevant edges from shared map points with other keyframe nodes. The spanning tree is then updated linking K_i with its parent node K_p which is the keyframe with the most shared map points. The BoW (4.2) representation of the keyframe is then stored in a database for use when tracking is lost, or relocalization.

4.4.2 Map Points Culling

Recent map points (observed from < 3 Keyframes ago) are checked for the following criteria, and culled if they fail them:

1. The tracking must find the point in $> 25\%$ of frames where it is predicted to be visible.
2. If more than one keyframe has passed from map point creation, it must be observed from at least three keyframes

4.4.3 New point creation

New map points are created by triangulating the tracked ORB features from connected keyframes K_c in the covisibility graph. Matches are searched for the unmatched points in K_i with points in frames of K_c, and if match is found they are checked to see if they satisfy the epipolar constraint (see appendix D.1). ORB pairs are triangulated, and to accept the new points,

4.4.4 Local Bundle Adjustment

The local Bundle Adjustment optimizes the current Keyframe K_i, all the Keyframes connected to it in the Covisibility graph K_c, and all the map points seen by those frames. Other keyframes that see those points but which $\notin K_c$ are included in the optimization but remain fixed. Details of the optimization are in 4.6.2

4.4.5 Local Keyframe Culling

The local mapping tries to detect redundant keyframes as frames in K_c whose 90% of the map points have been seen in at least other three keyframes in the same or finer scale, and delete these for reducing BA complexity, and bounding the number of local keyframes.

4.5 Loop Closing

Loop closing is an integral part used in many optimization based SLAM algoritns [69][70] which reduces global drift significantly. The loop closing thread takes K_i and tries to detect, and close loops. This is done in four steps which are described in the following subsections/

4.5.1 Loop Candidate Detection

- First, the similarity scores between the BoW representation of the query frame, K_i with its neighbors in the covisibility graph (for which min number of shared map points=30) is computed, and the lowest score among these is stored as s_{min}.
- Then the database is queried, and all Keyframes from the database for which the similarity score $s < s_{min}$ are automatically rejected. DBoW2 [82] uses a normalizing score to gain robustness, but ORB SLAM uses this covisibility score for robustness.
- All keyframes directly connected to K_i are also rejected for the loop closure candidates since these would generally have high scores, but not close a loop.
- To accept a loop candidate, three consecutive loop candidates that are consistent (keyframes connected in the covisibility graph) must be detected.

4.5.2 Transformation Computation

To estimate the error accumulated in the loop, and for a geometrical validation of the loop, the rigid body transformation between the current keyframe K_i and the loop keyframe K_l. ORB features associated with the map points in current frame, K_i and loop candidate frames K_l are obtained, and from this we get the 3D-3D correspondences from which the transformation can be computed with a closed form solution [68] and RANSAC. If enough inliers are found, the loop with K_l is accepted.

4.5.3 Loop Fusion

The first step for the loop correction is to is to fuse duplicated map points and insert new edges in the covisibility graph that will attach the loop

closure. The current keyframe pose T_{iw} is corrected with the transformation T_{il} and this correction is propagated to all the neighbors of K_i, concatenating transformations, so that both sides of the loop get aligned.
All map points seen by the loop keyframe and its neighbors are projected into K_i and its neighbors, and matches are searched in a narrow area around the projection. All those map points matched and those that were inliers in the computation of the transformation in 4.5.2 are fused.

4.5.4 Graph Optimization

The *Essential Graph* is optimized with a pose-graph optimization (4.6.3) that distributes the loop closing error along the graph. In the monocular case, the optimization is performed over similarity transformations to correct the scale drift, but in the RGBD case there is no need for this as we have the scale from the sensor input; and the optimization is hence over rigid transformations.

After the pose-graph optimization a full BA (4.6.2) is performed in a separate thread. However, if a new loop is detected while the optimization is running, the optimization is aborted and the looop is closed with the steps detailed above, which also again triggers a full BA.
When this full BA finishes, the updated set of Keyframes and points are merged with the non-updated ones by propogating the correction of updated frames to non-updated frames through the spanning tree. Non-updated points are transformed according to the correction applied to their referece Keyframe.

4.6 Optimization Problems

The different optimization and bundle adjustment problems used in the above sections are detailed here.

4.6.1 Motion-only Bundle Adjustment

The optimization variables are camera orientation $R \in SO(3)$ and translation $t \in \mathbb{R}^3$. The objective function is the reprojection error between 3D points $X^i \in \mathbb{R}^3$ and keypoints $x_s^i \in \mathbb{R}^{\mathbb{3}}$ consisting of u_l, v_l, u_r where $i \in$ set of all matches.

$$R, t = \underset{R,t}{\arg\min} \sum_i \rho(||x_s^i - \pi(R, t, X^i)||_\sigma^2) \tag{4.2}$$

where ρ is the robust Huber cost function [63], σ the covariance matrix associated to the scale of the keypoint, π a projection function where $\pi(X) = [\hat{u_l}, \hat{v_l}, \hat{u_r}]^T$, where the first two coordinates are obtained from the pinhole projection (appendix A) and the last coordinate u_r from (4.1).

4.6.2 Local Bundle Adjustment

The local Bundle adjustment optimizes a set of covisible keyframes K_L, and all points seen in those keyframes P_L. All other keyframes $K_F \notin K_L$ but which observe points in P_L contribute to the cost function but are not variables for the optimization, i.e. they remain fixed. Consider M_k the set of matches between points in P_L and 2D keypoints in a Keyframe k, we have the following problem:

$$X^i, R_l, t_l | i \in P_L, l \in K_L = \underset{X^i, R_l, t_l}{\arg\min} \sum_{k \in K_L \cup K_F} \sum_{j \in M_k} \rho(E_{kj}) \tag{4.3}$$

where

$$E_{kj} = ||x_s^j - \pi(R_k, t_k, X^j)||_\sigma^2 \tag{4.4}$$

Full BA. is a case of the above optimization problem where all keyframes and map points are optimized together, except the origin keyframe which is fixed at Identity.

4.6.3 Pose-Graph Optimization

The pose-graph optimization is a rough approximation of full BA, done with the Essential graph. Given an essential graph (4.3.2) , the error in an edge is defined as:

$$e_{i,j} = log_{SE(3)}(T_{ij} T_{jw} T_{iw}^{-1}) \tag{4.5}$$

where T_{ij} is the relative rigid body transformation estimated in 4.5.2. In the case of loop closure the value of $T_{ij} T_{jw} T_{iw}^{-1}$ should be Identity, since the transform is from $i\ frame$ $\rightarrow$ World frame $\rightarrow$ $j\ frame$ $\rightarrow$ back to $i\ frame$ The objective for minimization is:

$$C = \sum_{i,j} (e_{i,j}^T \Lambda_{i,j} e_{i,j}) \tag{4.6}$$

where $\Lambda_{i,j}$ is the information matrix of the edge, which is set to Identity in the ORB SLAM paper.

Chapter 5

Modifications to couple the network with ORB SLAM

This chapter details the engineering modifications to the network, and to the ORB SLAM code to enable the densification of the sparse point cloud from SLAM.

5.1 Modifications to the Network

The primary modification on the network side, is for the network training stage. The implementation by [1] uses a pose-network to directly regress the relative pose between frames needed to compute the Photometric error (3.3) which is trained simultaneously along with the depth completion network. While the same mechanism could be adopted to use with Keyframes, and sparse point clouds from ORB SLAM, it was found during experimentation (7) that the training converges faster, and performs better on the evaluation metrics when trained using the camera poses from the ORB SLAM tracking instead of the pose net. Most SLAM or VO algorithms, including ORB-SLAM track all the camera poses with respect to some global frame, usually initialized as the coordinate frame of the initial frame. To convert from the global camera poses, to the relative poses the following is done:

For three frames I_{t-1}, I_t, I_{t+1} with some absolute poses p_{t-1}, p_t, p_{t+1} respectively, which transform points from the world frame to the respective camera frame (see interpretation (C.3)), we need the relative poses, or the transforms $p_{\tau t}$ which transform a point in the frame of camera I_t to that of the frame of camera I_τ for $\tau \in T : \{t-1, t+1\}$.
Consider a 3D point X_{γ_t} in the frame of camera I_t. Our $p_{\tau t}$ must satisfy the following:

$$\begin{bmatrix} X_{\gamma_\tau} \\ 1 \end{bmatrix} = p_{\tau t} \begin{bmatrix} X_{\gamma_t} \\ 1 \end{bmatrix} \tag{5.1}$$

where X_{γ_τ} is the coordinate of the same point in the frame of I_τ.We get this p_{τ_t} as the composition of transformations (see C)

First the transformation p_t^{-1} transforms a point X from the frame of camera

I_1, γ_t to theworld frame δ as (see C):

$$\begin{bmatrix} X_\delta \\ 1 \end{bmatrix} = p_t^{-1} \begin{bmatrix} X_{\gamma_t} \\ 1 \end{bmatrix} \tag{5.2}$$

Then the transformation p_τ transforms the point X_δ from the world frame to the camera frame of I_τ. γ_τ.

$$\begin{bmatrix} X_{\gamma_\tau} \\ 1 \end{bmatrix} = p_\tau \begin{bmatrix} X_\delta \\ 1 \end{bmatrix} \tag{5.3}$$

From (5.2) and (5.3), we see that

$$\begin{bmatrix} X_{\gamma_\tau} \\ 1 \end{bmatrix} = p_\tau p_t^{-1} \begin{bmatrix} X_{\gamma_t} \\ 1 \end{bmatrix} \tag{5.4}$$

transforms the point X_{γ_t} from the frame of camera of I_t to the point X_{γ_τ} in the frame of camera of I_τ. We see that this satisfies equation (5.1) and we have the relative pose:

$$p_{\tau t} = p_\tau p_t^{-1} \tag{5.5}$$

In some cases, where p_{t-1}, p_t, p_{t+1} are the transforms from the respective camera frame to the world frame , the inverse of the 4×4 matrices are taken in the beginning, and the rest of the procedure remains the same. With this, we get the relative pose $p_\tau t$ for $\tau \in t-1, t+1$, which is used in 3.3.1 to compute the photometric error which the network is trained on.

Apart from this, there are multiple code modifications in the codebase of [1], for compatibility with the ORB SLAM data; some of them are:

- ORB SLAM saves the depth data (sparse and raw), as well as pose data as yml matrices instead of depth images.
- An artefact of the code in the original codebase made it such that though a triplet image is required for just the each training sample, they use the triplet image as inputs for the inference as well. This is undesirable since if we have a single image with sparse data we'd still like to run inference, and the changes to enable this were implemented.
- Modifications to enable inference of a single instance of image and sparse data were implemented; the original codebase could only run inference for testing/validation on a batch of data.
- Convenience scripts for data pre-processing, 3D visualization, etc.. were implemented.

5.2 ORB SLAM modifications

The ORB SLAM implementaion uses the following data structure objects, some of which contain overlapping information to represent various aspects described in chapter 4:

1. Frames
2. KeyFrames
3. KeyFrame Database
4. Map point
5. Map

The original datastructure objects, and their modifications are detailed. Not every variable, and function is detailed, but the most important ones to understand the overall functioning, and those that enable the changes possible are.

5.2.1 Frames

For every input RGBD data, a *Frame* object is created which originally contained the following:

- The Vocabulary for the Bag of Words representation (see 4.2)
- An orb extractor
- The intrinsic, and distortion parameters for the RGB camera
- Stereo baseline if stereo inputs, and imaginary stereo rig baseline to create the stereo points as detailed in 4.3.3 if RGBD .
- A vector of undistorted detected 2D ORB keypoints $mvKeysUn$
- A vector of the 'right' stereo coordinate of the keypoints from above $mvuRight$.
- A vector of the depths corresponding to those keypoints as input from the sensor. $mvDepth$
- A vector of the Bag of Words representation $mBowVec$, and a feature vector $mFeatVec$
- A vector of map points associated to keypoints in the frame $mvpMapPoints$
- The computed camera pose mT_{cw}
- Its reference Keyframe, i.e. the keyframe with which it shares the most map points $mpReferenceKF$
- Various functions to extract or compute all the above mentioned data, and also miscellaneous functions on the data.

To this are added the following:

- The raw RGB image. Though the original orb slam has no necessity for storing the image itself, the RGB image is required for use with the network
- The depth image from the sensor. The original algorithm just uses the depth data to convert to stereo coordinates with a baseline, but the depth data as an image is stored in this work to use as ground truth depth for a frame for evaluation purposes.

Although storing the images is not efficient memory-wise it is imperative that there is access to this data so that they can be stored, and used with the network for training, inference, and evaluation.

5.2.2 KeyFrames

The Keyframes are 'special' frames which are the nodes in the covisibility graph described in chapter 4. They contain the following information:

- A Keyframe ID, the frame ID, and the ID of the next Keyframe
- local Bundle Adjustment (see 4.6.2) variables
- Variables for loop closure, and re-localization
- Intrinsic, and distortion parameters carried over from the Frame data structure.
- vectors of Keypoints, stereo coordinates, and their descriptors
- The Bag of Words vectors
- Pose of Keyframe T_{cw}
- Inverse pose of Keyframe, i.e. pose of the keyframe system in the world coordinate frame T_{wc}.
- The camera center in the world frame, O_w
- The vector of Map points associated to the keyframe $mvpMapPoints$
- Variables for the spanning tree, and Loop Edges
- Thread locked function to get the map points associated with the keyframe. Since it is thread locked, the map points cannot be obtained from a function call to this function outside the thread in the main program, and hence it is not possible to get the map points' world coordinates from a KeyFrame.
- Various other functions to extract or compute the above mentioned data, and also miscellaneous functions on the data.

Similar to the Frames, the KeyFrames data structures are also made to store the RGB data for inference with sparse depth, and the raw depth image for evaluation. These are propogated to the Keyframes from Frames.

5.2.3 KeyFrame Database

The KeyFrame database, as the name suggests stores the KeyFrames, with their bag of words representations for loop detection, and relocalization. This work does not use this data structure object, and there are no changes to it.

5.2.4 Map Point

The Map point object, as the name suggests stores information about a point in the created map. It has the following members:

- The ID of the point, $mnId$
- The IDs of the first KeyFrame ID, and the first Frame ID.
- number of times the point is observed $nObs$
- Variables used by the tracking
- Variables used by the local mapping, which optimizes the map points and poses
- Variables used by the loop closing
- The position of the point in absolute coordinates (in the World Frame), $mWorldPos$
- An std::map of Keyframes observing the point and associated index in the keyframe, $mObservations$
- The reference keyframe for the point, $mpRefKF$
- The Map object of the entire map
- Various functions, and other variables.

The map point is an important object which is used in the main program, outside any of the three threads (tracking, local mapping, loop closure) to get the 3D World coordinates of a point.

5.2.5 Map

The data structure object storing a representation of the entire map. It consists of:

- A vector of the Keyframe origins of the points in the map
- An std::set of all the map point objects, $mspMapPoints$
- An std::set of all the KeyFrame objects associated with any point in the map, $mspKeyFrames$
- A vector of reference map points, $mvpReferenceMapPoints$
- Functions related to map manipulation, and to get information about the map objects.

5.2.6 Main program

In the main program where all the submodules are run, the data we require is accessed, checked, and saved while a function to save the map is called. For training the network on Orb SLAM data, we need the following:

1. RGB Images
2. Sparse depth data from the map points, corresponding to the RGB images in the frame of the cameras
3. Validity maps which are boolean images with $True$ value for valid depths, and $False$ for invalid ones in the sparse data.
4. Raw depth from the sensor corresponding to each frame for evaluation. This is not used during training
5. Pose data for each frame used for the training and reprojection.

As mentioned in 5.2.2, we cannot get the world coordinates of the map points from a mappoint obtained from a KeyFrame object in the main program. Hence, to obtain the depth, we extract all the map points together from the map object, extract the 3D coordinates from them, and project them into each keyframe by which they are observed. This information is created, and extracted with the following procedure:

- First, from the map datastructure object (5.2.5) we use one of its functions $GetAllKeyFrames()$ to get all keyframes for points in the map and store it in a vector of KeyFrames(5.2.2) $vpKFs$ which are sorted according to their IDs.
- Then, using the function $GetAllMapPoints()$ from the map datastructure object, we get the vector of map points(5.2.4)
- Looping over the vector of map points mps:
  - The 3D coordinates of the map point with respect to the world frame X_δ is extracted with the function $GetWorldPos()$.
  - We then loop over every keyframe in the vector $vpKFs$. For Keyframe K_j for $j \in length(vpKFs)$:
    - First, check if the keyframe observes the map point. If it does not, iterate to the next Keyframe.
    - Extract the vector of 2D ORB keypoints $mvKeysUn$ of KeyFrame K_j.
    - Extract the intrinsic camera matrix parameters from K_j, and construct the matrix K for KeyFrame K_j
    - Then, the index of the map point in the KeyFrame is extracted. This index refers to the 2D keypoint in the KeyFrame, to which the map point corresponds.

  - With this index, the 2D image coordinates of the keypoint (u, v) from *mvKeysUn* corresponding to the map point is extracted.
  - The pose p, of the form (C.1), of KeyFrame K_j is extracted.
  - With the pose p, and the intrinsic matrix K, we obtain the image coordinates of the projection $(\hat{u}, \hat{v})$ of X_δ (see A)
  - The reprojection error is computed as:

$$r = || \begin{bmatrix} \hat{u} \\ \hat{v} \end{bmatrix} - \begin{bmatrix} u \\ v \end{bmatrix} ||_2 \tag{5.6}$$

  - If the reprojection error $r < error\ threshold$, then the z coordinate of the point X in the camera coordinate frame. i.e..

$$\begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} = p \begin{bmatrix} X_\delta \\ 1 \end{bmatrix} \tag{5.7}$$

    and depth $d = z$
  - The coordinates of the 2D keypoint (u, v) along with the depth value d of the map point, and the pose p_j for the Keyframe K_j is stored in some respective variables.

- This is used to create the sparse depth map for each keyframe, the rgb image and raw depth is extracted from the KeyFrame object, and the validity map is created as an image which has values of 1 in pixels where the sparse depth > 0, and 0 otherwise.

5.3 Combining ORB SLAM data with KBNET network

Once the data from ORB SLAM described in the previous section is available, it is set up in a format compatible with the KBNet training and inference (described in chapter 7). The network is then trained to overfit on this training data since we wish to densify the same data. The reason for this approach is elucidated in chapter 7.
Once we have the best model from validataion (a subset of the training data in this case), inference is run for all the images with their respective spare depths and intrinsic matrices, obtaining the output dense depth maps.

5.3.1 Dense depth back-projection

For a selection of the keyframes, we back-project these dense depths to the original pointcloud from ORB SLAM by the following:

Given a keyframe i, we have the RGB image I^i, sparse depth D^i obtained as described in 5.2.6, for intrinsic matrix K we get the estimated depth $\hat{D}$

from running the inference. We also have the pose p_i from ORB SLAM for the keyframe. For a point j with some pixel coordinates (u^i, v^i), the corresponding estimated dense depth value is then d^i_j. To get the 3D coordinates X_{γ_i} in the frame of camera i, we follow the process shown in 3.3.1. Direction vector

$$\vec{x^i_j} = \lambda_i K^{-1} \begin{bmatrix} u^i \\ v^i \\ 1 \end{bmatrix} \tag{5.8}$$

$$X_{\gamma_i} = d^i_j \vec{x^i_j} \tag{5.9}$$

We get the coordinates of the point in the world frame, X_δ with:

$$\begin{bmatrix} X_\delta \\ 1 \end{bmatrix} = p_i^{-1} \begin{bmatrix} X_{\gamma_i} \\ 1 \end{bmatrix} \tag{5.10}$$

We use the property that p can be seen as a transformation from world frame to camera frame, its inverse p^{-1} is the transformation from camera frame to the world frame.(C.3). This is done for every point j in keyframe i, and for every i in the list of keyframes to be backprojected.

Chapter 6

Data collection and Dataset creation

The ARI robot is a humanoid platform designed for a wide range of multimodal expressive gestures and behaviors, suitable for Human-Robot-Interaction, perception, cognition and navigation. It provides an interesting case where some form of depth completion is almost necessary for functioning of the robot since navigation often runs into collisions for areas which were not sufficiently mapped by a sparse mapper. For this reason, testing on data captured from the actual robot is vital to gauge its validity. The robot has multiple cameras for mapping, localization, navigation and sensing the environment; the camera configuration in the robot is shown in figure 6.1.

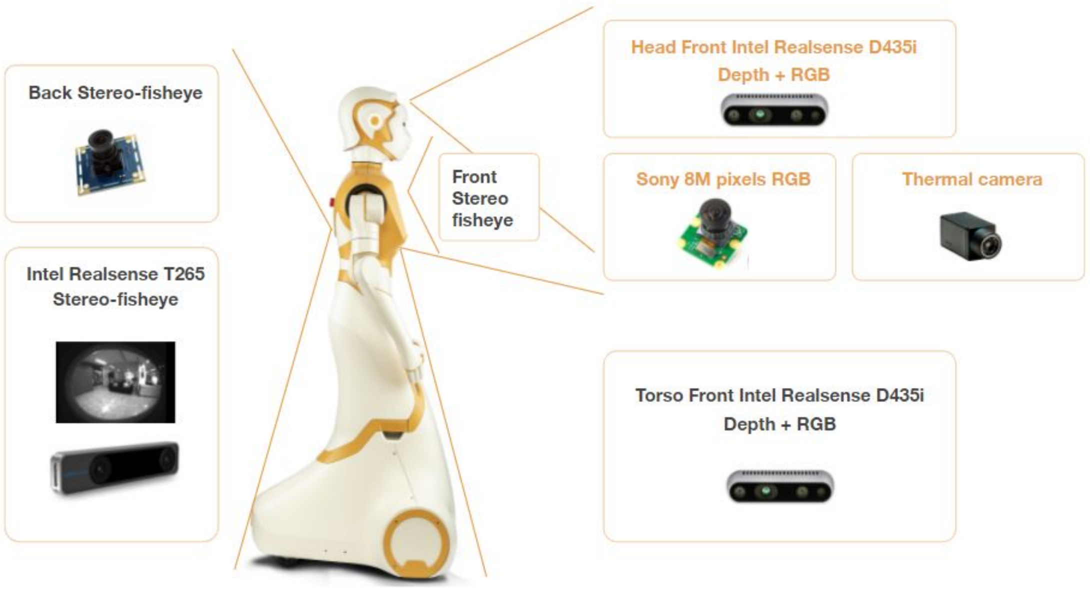


Figure 6.1: ARI Cameras Configuration. The Torso Front camera, an Intel Realsense D435i is used with ORB SLAM

The torso front camera, which is an Intel Realsense D435i RGBD camera is used for the data collection, and dataset creation in this work. The Realsense D435i is an RGBD camera consisting of an RGB imagine module, along with two monochrome cameras in a stereo rig set-up, and an IR projector, along

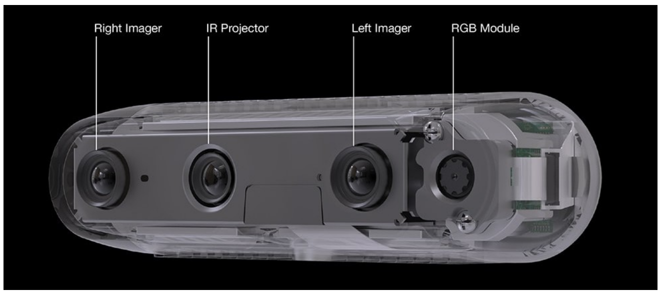


Figure 6.2: D435i camera module. Consists of an RGB camera, and 2 monochrome cameras in a stereo rig, along with an Infraeed projector

with an Inertial Measurement Unit (IMU). The IR projector is used as an active stereo imaging technique to project an IR pattern into textureless surfaces which is detected by the monochrome cameras. However, in this work this IR projector is turned OFF since the challenge is to estimate depth in areas of low/no texture without an active component.

The camera is pre-calibrated, and the intrinsic matrix parameter values are provided by the manufacturer. The RGB camera is operated at 848×480 resolution at 30 frames per second for the image, as well as depth data. The depth is obtained from the stereo rig consisting of the two monochrome cameras and the IR projector. We use the RoboticOS(ROS) [85] framework to work with the camera, as well as the robot as a whole. The data is stored in the form of a *rosbag* containing the RGB camera information, aligned depth information, and the camera intrinsics, and distortion parameters with the topics:

1. $/torso\_front\_camera/color/image\_raw/compressed$: containing the RGB frames in the form of $sensor\_msgs/compressedImage$ messages
2. $/torso\_front_camera/aligned\_depth\_to\_color/image\_raw$: containing the depth aligned to the RGB camera frame in the form of $sensor\_msgs/Image$ messages. Each pixel in a frame of this topic contains the scaled depth value corresponding to that pixel location in the RGB image with a scale factor of 1000.
3. $/torso\_front\_camera/color/camera\_info$: containing the camera information, i.e. the parameters of the intrinsic matrix K, and the distortion parameters for the camera. In our case, the RGB image we obtain is rectified (undistorted) by the ROS wrapper, so we can assume 0 distortion.

The open-source ROS wrapper for ORB-SLAM2 works with raw RGB images, and raw aligned depth as inputs. Since recording raw images is not convenient for lengthy recordings, functionality to synchronize and work with compressed images and raw aligned depth, as well as compressed images, and compressed aligned depths had to be added. The dataset consists of the recorded rosbags, and the results of the ORB-SLAM using 5.2. The structure

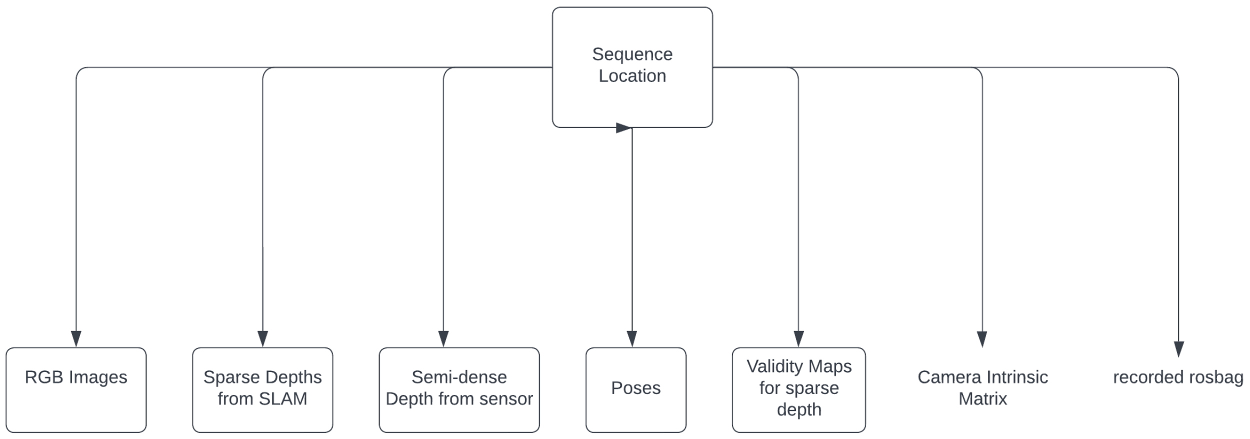


Figure 6.3: Dataset structure after running the modified ORB SLAM on the data from the recorded rosbags. Each sub-folder under the sequence location directory contains files (images or 'yml's) for each keyframe in the SLAM. The camera intrinsic matrix is stored as a numpy 'npy' matrix.

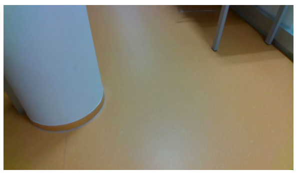
for each sequence in the dataset is shown in figure 6.3. The RGB images, and validity maps for the sparse depth are stored as 'png's while the sparse depth from SLAM, as well as the semi-dense depth which we use as ground truth for evaluation are stored as 'yml's, and the Camera intrinsic matrix file is stored as a numpy 'npy' matrix.

The data was recorded from indoor locations in CIIRC building, as well as in a hospital. The robot was controlled with a joystick, and moved at slow translational and rotational speeds in a manner which ensures that the ORB SLAM would not lose tracking. Additionally, since ORB SLAM performs a pose-graph optimization (4.6.3), and full bundle adjustment on loop detection and closure as described in section 4.5, the robot was moved in a looped trajectory which would ensure that the loop closure was triggered. The data was recorded in the following locations:

1. A small room #637 in CIIRC
2. A relatively larger room AAG impact room in CIIRC
3. A large lecture hall #670 in CIIRC.
4. A corridor and hallway at BROCA hospital in Paris, for SPRING.

The rosbags for the data in the BROCA hospital was recorded with not only the topics listed previously, but also various other data required for the SPRING project. Due to this, the sequence contains multiple bags of large size, and the aligned depth data as well as the RGB data are both compressed.
Each sequence has challenging images, both due to the nature of the downward facing camera itself, and meeting pillars, and other textureless objects in the

(a) : Image from Room 637


(b) : Image of a white pillar from Room 670

(c) : Image from the BROCA hospital hallway

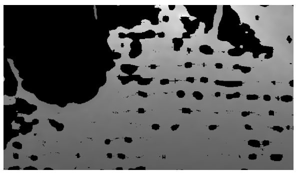

(d) : Ground truth of (b)

Figure 6.4: Examples of some images from different sequences in the dataset. Image (d) shows the ground truth obtained from image (b), where we see that almost no points from the pillar are in the 'ground truth'

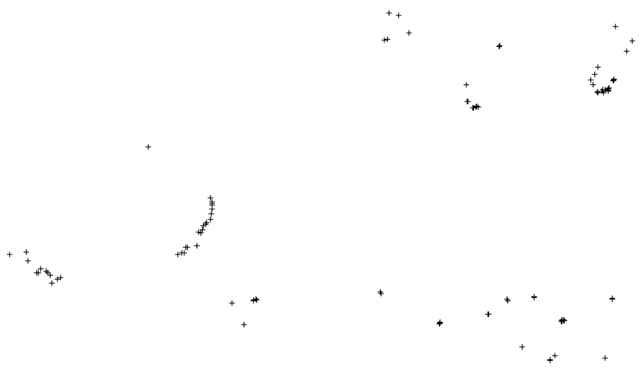


Figure 6.5: Sparse depth image from SLAM corresponding to img (b).The image has been inverted and the sparse points dilated to appear larger than they actually are; real sparse data is even sparser than is shown in the image 6.4b;

path. Some examples are shown in figure 6.4. For most images, the 'ground truth' from the sensor is incomplete in varying degrees since the depth values are obtained from stereo, and stereo relies on textured surfaces as well. An example of ground truth corresponding to figure 6.4b is shown in figure 6.4d. The sparse data which we obtain from points tracked in ORB SLAM contains about few hundred points per frame, which is significantly lower than in other datasets which contain thousands to tens of thousands of points per frame. This adds to the complexity of the task and provides an additional reason as to why a previously trained model on other datasets can be employed. An example of the sparse depth image is shown in figure 6.5. We see that there are a few points near the base of the pillar, but no points along the pillar itself.

Chapter 7

Implementation, Experiments and Results

The implementation details such as description of hyperparameters for the network training and inference, ORB SLAM parameters, steps for dataset preprocessing, as well as the experimentations and their results are described in this chapter. Experiments were carried with the VOID dataset [4], as well as the captured data described in chapter 6

. The KBNet model is implemented in PyTorch [86] using the Adam optimizer [84] with the initial decay rates hyperparameters $\beta_1 = 0.9$, and $\beta_2 = 0.999$ to optimize the network. [1] trains with a batch size of 8 for 15 epochs on the VOID dataset. Since the data from a single ORB SLAM sequence is considerably smaller, we use a batch size of 4, and significantly more epochs to overfit. Results pertaining to these hyperparameters are presented in later sections in this chapter.

NOTE: The units for the evaluation metrics are as follows: MAE, and RMSE are in mm and the iMAE, and iRMSE in mm^{-1}

7.1 Experiments and Results on the VOID dataset

The VOID dataset is described in section 2.1. The dataset we create with the robot data also mimics this dataset. This section details some experiments and their results on the network with the VOID dataset. For this, we use a subset of the void dataset consisting of a total of 36451 samples with 35917 samples in the training set, and 534 samples in the testing and validation sets. Mimicking [1], the training is validated on the same set of samples as the testing set. The evaluation metrics used are the Mean Absolute Error (MAE), Root Mean Squared Error (RMSE), inverse Mean Absolute Error (iMAE),and the inverse Root Mean Squared error (iRMSE) (see section 2.1), and the units are in mm or 10^{-3}m

The weights for the Loss function are initially set to $w_{ph} = 1$, $w_{co} = 0.15$, $w_{st} = 0.95$, $w_{sz} = 2$, and $w_{sm} = 2$. The meaning behind these weights is explained in section 3.3. The kernels for the min, and max pooling operations in the Sparse-to-Dense Module (3.1) is 3 min-pooling layers of sizes $[15, 17, 19]$ and 2 max-pooling layers with kernel sizes $[23, 27]$. The learning schedule for the 15 epochs are split as 1×10^{-4} for the first 10 epochs and 5×10^{-5} for the last 5. Data augmentations such as randomly removing 30% to 60% of the sparse points are enabled 100% of the time, following [1]. The step where the validation loss is the least is chosen as the 'best' model from the training.

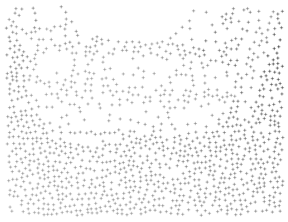
Training with and without a pose-network. Though the VOID dataset contains poses corresponding to every image, and sparse depth data, [1] do not use these poses for their training, nor do they test the difference between using the Pose network, and using these poses. Modifying the training mechanism to use the poses (see 5.1), the training using posenet outperformed that with poses. The quantitative results are presented in table 7.1, and the qualitative visualizations are shown in figure 7.1. For the visualization, all the depths are normalized to lie in a range of 0-255 where 0 corresponds to the minimum value, and 255 to the maximum value. The normalized pixelwise L_2 difference between the predicted dense depth, and the ground truth depth is visualized as an inverse difference image.
From figure 7.1 we see that the prediced depths do not differ significantly from either method in most areas, but the model trained on absolute poses performs worse in the pillar as can be seen in the inverse difference images. While in the case of the VOID dataset training with posenet outperforms using the poses quantitatively, we will see that it is not so for the data captured from the robot. A possible reason for this could be that the poses of the VIO used in VOID dataset are outperformed by the Posenet, and since the photometric loss (3.3) depends heavily on them, they would affect the model accuracy.

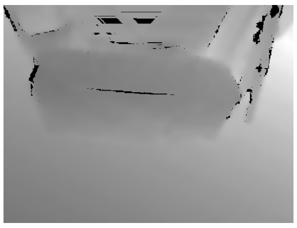
Table 7.1: Results on the testing set of VOID dataset [4] for the network trained on relative poses from 1.Posenet, and 2.Absolute poses from ORB SLAM [2]. Training PoseNet simultaneously and using it for the relative poses seems to give better results in the case of the VOID dataset

Pose source	MAE	RMSE	iMAE	iRMSE
Relative poses from PoseNet	31.294	79.999	16.512	39.643
Absolute poses from VIO	35.596	89.272	20.327	46.497

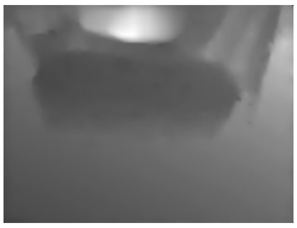
Training with different loss function weights. The loss function weights were experimented with the model trained using the absolute poses. The results are presented in table 7.2. The best results were obtained with the values used in [1]. Moreover, when the structural loss weights were increased thus giving it more importance, the validation loss got worse with training epochs, which again suggests that the absolute poses in the dataset might not be contain inaccuracies.

(a) : RGB Image from the VOID dataset


(b) : Sparse depth image with 1500 points


(c) : Ground Truth depth for the image

(d) : Predicted dense depth using the model trained with poses from PoseNet


(e) : Predicted dense depth using the model trained with absolute poses

(f) : L_2 difference image (inverted) visualized for model trained with posenet. Brighter is more difference, and darker is less

(g) : L_2 difference image (inverted) visualized for model trained with posenet.

Figure 7.1: Visualizations of outputs of models trained using poses from a pose-network, and with poses from absolute poses

Table 7.2: Results on the testing set of VOID dataset [4] for the network trained on different loss function weights, and with absolute poses

w_{ph}	w_{co}	w_{st}	w_{sz}	w_{sm}	**MAE**	**RMSE**	**iMAE**	**iRMSE**
1	0.15	2.0	2.0	1.0	46.346	101.458	28.349	58.658
1	0.15	2.0	1.5	1.5	54.043	109.192	31.220	61.149
1	0.15	3.0	2.0	1.0	90.494	137.657	48.679	76.186
1	**0.15**	**0.95**	**2.0**	**2.0**	**35.596**	**89.272**	**20.327**	**46.497**

7.2 Experiments and Results on data captured from the robot

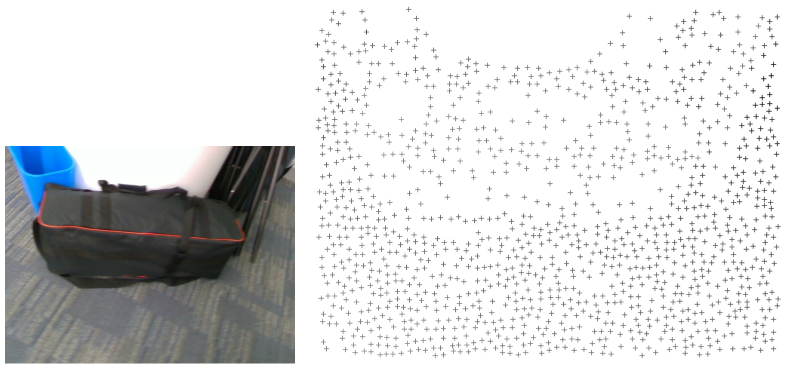
Dataset Preprocessing. Since the training of the network (3.3) uses 3 images per training sample, in [1] they concatenate three images I_{t-1}, I_t, I_{t+1} into one combined image which is loaded along with the sparse depth for S_t, and the intrinsic matrix K_t, and the relative poses $p_{\tau t}$ for $\tau = \{t-1, t+1\}$ regressed from the pose-network. In this work, the same concatenation is performed on the image data resulting in a single $2544 \times 480 \times 3_{channes}$ image stacked along the width from three $848 \times 480 \times 3_{channels}$ images for each training sample; the pose data with respect to some global frame are also concatenated to a 12×4 matrices stacked along the rows. These poses are separated into four 4×4 matrices while reading and converted to relative poses during training using the method described in section 5.1. This concatenation of poses is done solely for easier integration with the existing codebase.

Data split. For the data captured with the robot, the data split is different from the previous section, since in this case, our aim is to train and run inference on the same data. Hence, all the samples from a sequence are used for training and testing. For validation, we take a random subset of this consisting of 0.1 times the size of the data.

Depth Scaling. A vital implementation difference, which causes problems if missed, between the VOID dataset, and the data we capture obtained from ORB SLAM is that; the VOID dataset [4] scales the depth by a factor of 256 to preserve the floating point accuracy. However, when the depth data is obtained from ORB SLAM, this is not done since the values are stored without losing their floating point accuracy as 'yml' files.

Testing with pre-trained model on VOID. First, inference is run on the data captured from the robot from Room 637 in CIIRC, and lecture hall 670, with the best performing model on the VOID dataset, i.e. the model trained with relative poses from posenet whose results are presented in table 7.1. The evaluation on this produces, as expected, large errors. This is because

- Though the VOID dataset is captured using the same camera and the model takes in the Intrinsic matrix as an input, the images are at a lower resolution of 640×480 in VOID, whereas they are 848×480 from the robot.
- The sparsity from ORB SLAM tracking is much worse, and unevenly distributed. In the VOID dataset, the sparse points are sampled using a corner detector with a low threshold, and then clustered to ensure that they are sampled from throughout the image, and not concenrated in some areas with constant density of 0.15%, i.e. 1500 points per image. Opposed to this, ORB SLAM tracks keypoints only based on the FAST [66] detectors, and the tracking, which results in an uneven distribution


(a) : RGB image, and Sparse depth form VOID

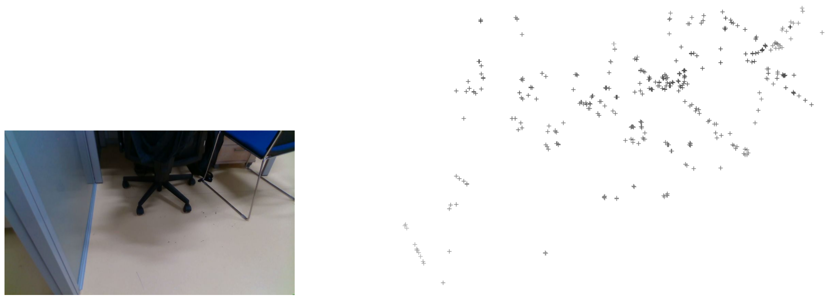

(b) : RGB Image from the robot and Sparse depth image from ORB SLAM

Figure 7.2: Difference in nature of data between the VOID dataset and the data captured from the robot. Sparse depth rescaled for visualization

of points as well as much fewer points (between 100-1000) points in an image.

- The dataset is captured using a robot from a downward facing camera. Though the VOID dataset has different sequences in various environments, very few of its images are similar to those captured by the robot.

Due to this, using the pre-trained network trained on a datasets like VOID, or others wouldn't work. The results are presented in table 7.3 and 7.4

Results from the training on the captured data, and the training procedure is presented in this section. Results are presented for the training with poses obtained from the ORB SLAM as well as using the pose-network, and with different learning schedules.

Overfitting. As explained in previous sections, since the the densification process is performed as an offline step with all the data from a sequence available, the model is trained to overfit on the data so as to achieve the best results on it. Training on limited data for a large number of epochs, in practice, results in a model learning or 'memorizing' the parameters to map from images and sparse depth to dense depth.

The data was trained and tested on two sequences from the robot; the first being a small room Room 637, and the other a large room lecture hall 670. The first sequence contains 448 samples, with each consisting of: an image

Training	epochs	Tr min_k	Tr max_k	Ev min_k	Ev max_k	MAE	RMSE	iMAE	iRMSE
VOID posenet	15	[15,17,19]	[23,27]	[15,17,19]	[23,27]	1318.664	1354.608	5337.475	5602.716
ORB posenet	150	[15,17,19]	[23,27]	[15,17,19]	[23,27]	360.522	448.070	198.450	237.920
ORB ORB	150	[15,17,19]	[23,27]	[15,17,19]	[23,27]	78.956	125.638	39.758	111.631
ORB ORB	150	[15,17,19]	[23,27]	[25,27,29]	[33,37]	78.222	126.041	39.570	112.032
ORB ORB	150	[25,27,29]	[33,37]	[25,27,29]	[33,37]	76.779	116.763	39.985	94.212
ORB ORB	150	[25,27,29]	[33,37]	[15,17,19]	[23,27]	78.108	118.492	40.521	94.619
ORB ORB	150	[15,17,19]	[23,27]	[25,27,29]	[33,37]	78.222	126.041	39.570	112.032
ORB ORB	250	[15,17,19]	[23,27]	[15,17,19]	[23,27]	78.464	121.093	38.083	88.373
ORB ORB	500	[15,17,19]	[23,27]	[15,17,19]	[23,27]	**72.278**	**115.754**	**33.656**	**68.020**

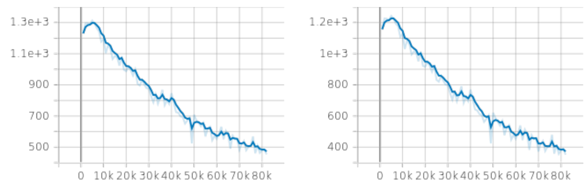
Table 7.3: Results on data collected from the robot, and sparse depth obtained from ORB SLAM sequence Room637. (Tr stands for training, Ev for evaluation, min_k and max_k for min and max pooling kernel sizes in the S2D module)

Training	epochs	Tr min_k	Tr max_k	Ev min_k	Ev max_k	MAE	RMSE	iMAE	iRMSE
VOID posenet	15	[15,17,19]	[23,27]	[15,17,19]	[23,27]	1337.092	1378.701	6443.715	6863.004
ORB posenet	150	[15,17,19]	[23,27]	[15,17,19]	[23,27]	351.743	453.680	181.208	213.572
ORB ORB	150	[15,17,19]	[23,27]	[15,17,19]	[23,27]	86.657	157.543	38.535	**68.639**
ORB ORB	150	[15,17,19]	[23,27]	[25,27,29]	[33,37]	126.924	199.744	54.955	81.512
ORB ORB	150	[25,27,29]	[33,37]	[25,27,29]	[33,37]	**83.948**	**157.052**	**36.758**	89.660

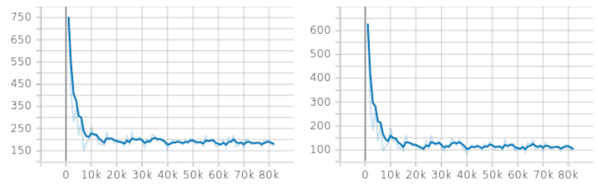
Table 7.4: Results on data collected from the robot, and sparse depth obtained from ORB SLAM sequence Lecture Hall 670.

triplet $\{I_{t-1}, I_t, I_{t+1}\}\}$, the pose triplet: $\{p_{t-1}, p_t, p_{t+1}\}$, the sparse depth s_z corresponding to I_t, and the second sequence contains 2202 samples. The loss function weights used for training are the same as the best ones for VOID shown in table 7.2; the training is done with batches of size 4.

Results. Since the sparse depth is significantly sparser in the ORB SLAM output data, than in the VOID dataset (see example in figure 7.2), as per the suggestion in [1], we use bigger kernels for the min and max pooling layers in the Sparse-to-Dense (S2D) module. The results show that this is indeed useful, and the evaluation metrics improve on both sequences. The results are presented in table 7.3 for the first sequence, and in table 7.4 for the second sequence. In examples shown in figure 7.4 and 7.5, it is seen that due to the nature of the scene, the ground truth depth itself is incomplete, or only semi-dense. This is because the ground truth is generated by stereo algorithms , and with the IR emitter in the camera turned OFF, it struggles to measure the depths in areas of no texture.



(a) : The RMSE, and MAE error metrics from the validation set during training with a pose-network

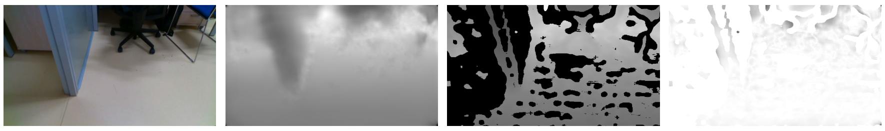


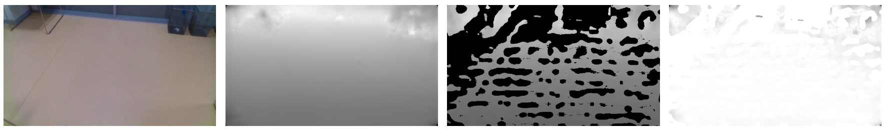
(b) : The RMSE, and MAE error metrics from the validation set during training with a pose-network

Figure 7.3: Validation error metrics through training steps for (a) network trained with a pose-network, and (b) network trained with poses from SLAM

While in both sets when the network is both trained and tested on bigger pooling kernels for the S2D, it performs outperforms a network trained on smaller pooling kernels by a little; however when the network is trained on smaller pooling kernels, and tested with larger kernels, the performance is nearly the same in the case of the first sequence, and is markedly worse for the second sequence. This highlights the importance of the Sparse-to-Dense module to the process. The training epochs affect the performance as well, although the difference is not significant. Hence, it might not be worth the time cost to train the extra epochs for a small performance boost.
Another noteworthy observation is that not only are the evaluation errors significantly higher when the network is trained with a pose-network, but the validation loss does not saturate to a minimum even with 150 epochs. This can be seen in the graphs plotting a validation error metric through training steps for the posenet, as well as with using ORB SLAM poses in figure 7.3.

Example visualizations of the network outputs are shown in figures 7.4 for sequence 1, and 7.5 for sequence 2. In the visualizations we see that the network predicts depths where even the ground truth fails to estimate them. The sparse points from the point cloud are not visualized since there are too few points to view at this image scale; this makes the performance of the network all the more impressive. The inverted difference images are visualized as well, where the L_2 difference between the predicted dense depth and the ground truth depth at each pixel are computed and normalized to a scale of [255-0] for visualization. Since the image is inverted, a value of 0 corresponds to the highest L_2 difference, and a value of 255 for 0 difference, i.e.. brighter a region is, the more relatively accurate it is. This gives us an idea of the relative error between areas in the image, but not the absolute error.


(a) : Example image, dense depth, ground truth, and inverted difference image 1


(b) : Example image, dense depth, ground truth, and inverted difference image 2

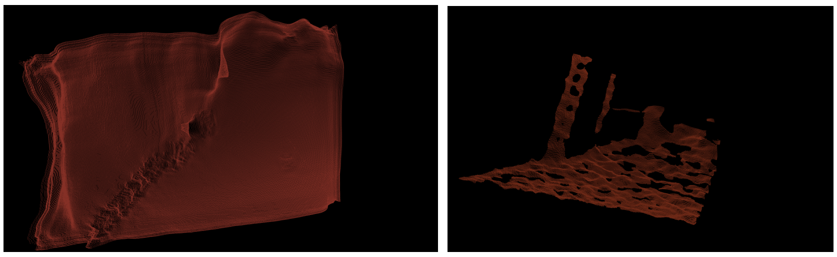
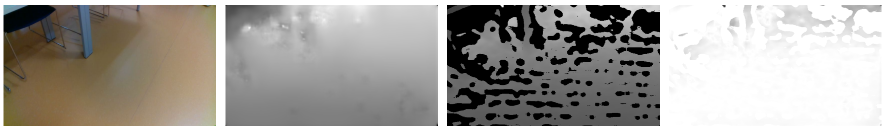
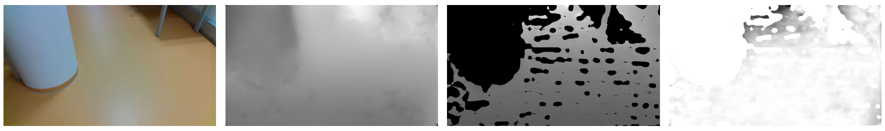

(c) : Pointclouds of projected predicted dense point cloud, and semi-dense ground-trugh for image (a)

Figure 7.4: Visualizations of the results for sequence 1 from the robot dataset. We see that the network predicts dense depth even in areas where the ground truth fails. We see that the network predicts dense depth even in areas where the ground truth fails.


(a) : Example image, dense depth, ground truth, and inverted difference image 1


(b) : Example image, dense depth, ground truth, and inverted difference image 2

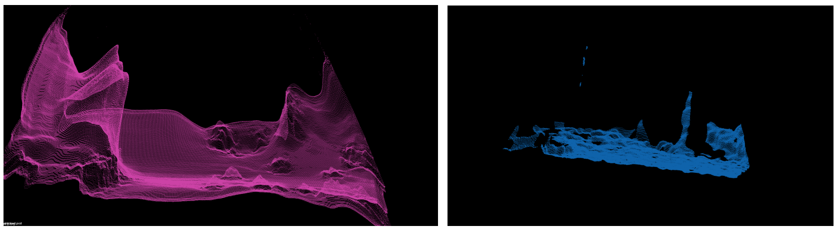

(c) : Pointclouds of projected predicted dense point cloud, and semi-dense ground-trugh for image (b)

Figure 7.5: Visualizations of the results for sequence 2 from the robot dataset. While the ground truth is entirely absent for the pillar in figure (b), the dense depth predicts depths in the region. The pointclouds show the projected points using the dense, and ground truth depths. The points from the dense depth for the floor are not completely flat, especially towards the edges of the image which is also reflected in parts of the inverted difference images.

Chapter 8

Conclusion and Future Work

8.1 Summary

This work presents

- A review on datasets, commonly used evaluation metrics, and systems for monocular depth completion and estimation with a compilation on available open-sources codebases in chapter 2.
- The workings of the KBNet network, and ORB-SLAM in chapters 3, and 4
- A method to densify a sparse scene representation obtained from SLAM [2] using a monocular depth completion network [1] in chapter 5
- Details setup for data capture, and results of experimentation from data on the ARI robot in chapters 6, 7

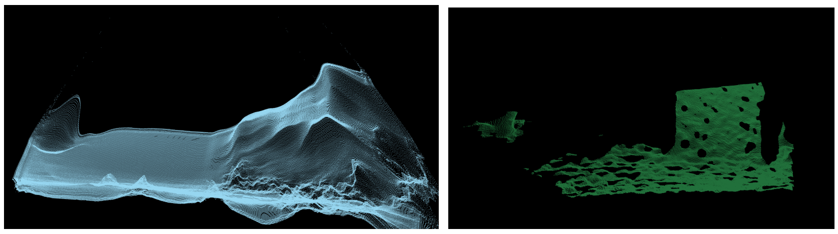
The densification process is currently carried out as an offline step after the sparse mapping, and can be used to create a dense representation of the scene in areas where the sparse mapping fails to capture any points, before navigation. The depth prediction network manages to use RGB images, with extremely sparse depths to produce dense depth maps even in areas where the ground truth from the sensor fails. This comes with the drawback that we have no way to quantitatively measure the errors from these regions, but rely on qualitative assessment.

8.2 Future Work

Selective Back-Projection. The results from section 7.2 show that the depth prediction manages to estimate depth even with extremely sparse inputs, with some exceptions, even in regions where the ground truth from a stereo-based RGBD sensor cannot. However, we cannot efficiently measure the accuracies of predictions in these problematic, but extremely important areas. Hence, rather than using the depth estimation predicted depths for every point of every frame, a focus of the future work should be on identifying certain 'problematic' key-frames, and specific areas in them for which we use the estimated depth to backproject. Estimating the uncertainity associated with the predictions [13][87][88] would also aid this selective back-projection, since we then discard predictions with high uncertainty.

(a) : Image, predicted dense depth, and the ground truth depth

(b) : Example image, dense depth, and the ground truth, where ground truth is missing for a large part of the image


(c) : Pointclouds of projected predicted dense point cloud, and semi-dense ground-trugh for image for image (a). We see from the point cloud of the predicted dense depth that the surfaces are 'bent' towards the camera at the edges

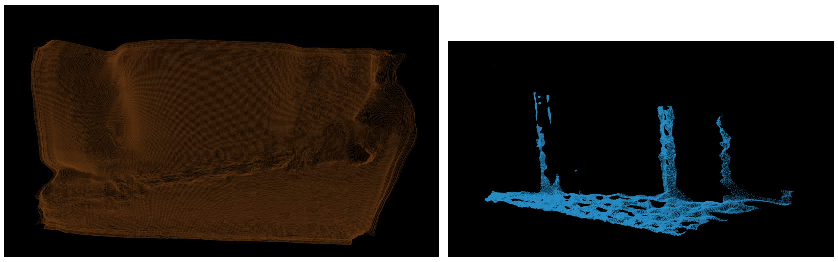

(d) : Pointclouds of projected predicted dense point cloud, and semi-dense ground-trugh for image for image (b). Here, the point cloud of the predicted dense depth seems more accurate, and the surfaces are flat

Figure 8.1: The need for an selective back-projection, and an uncertainty measure. While in the case of (d), the predicted depth 'seems' accurate when backprojected to the point cloud, in (c) we see that the surfaces aren't flat but bent

Online depth completion with SLAM. While the method outlined in this work is sufficient for systems which perform mapping and navigation asynchronously, many systems work in environments where the mapping and navigation take place together. This would require moving the depth completion from an offline step, to an online one, being performed in synchronization with the mapping, for which the current training regime would hence not be applicable. Ground robots also use 2D occupancy grids probabilistically mapped from the 3D SLAM data for navigation. While there are techniques to convert 3D point clouds to occupancy grids, the probablistic method involves observing multiple observations of a scene which can only be done online. An added advantage with optimization-based methods is that while running the algorithms online, they optimize on all available points to minimize reprojection errors or to close loops with pose-graph optimizations which is not done with the back-projected points in the offline case. Hence, some continual learning techniques [90] can be incorporated to 'learn as we go' and continually improve the system with more data to enable these.

Bibliography

[1] Alex Wong and Stefano Soatto. Unsupervised depth completion with calibrated backprojection layers. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 12747–12756, 2021.

[2] Montiel J. M. M. Mur-Artal, Raúl and Juan D. Tardós. ORB-SLAM: a versatile and accurate monocular SLAM system. *IEEE Transactions on Robotics*, 31(5):1147–1163, 2015.

[3] Raul Mur-Artal, J. M. M. Montiel, and Juan D. Tardos. ORB-SLAM: A versatile and accurate monocular SLAM system. *IEEE Transactions on Robotics*, 31(5):1147–1163, oct 2015.

[4] Alex Wong, Xiaohan Fei, Stephanie Tsuei, and Stefano Soatto. Unsupervised depth completion from visual inertial odometry. *IEEE Robotics and Automation Letters*, 5(2):1899–1906, 2020.

[5] Chaoqiang Zhao, Qiyu Sun, Chongzhen Zhang, Yang Tang, and Feng Qian. Monocular depth estimation based on deep learning: An overview. *CoRR*, abs/2003.06620, 2020.

[6] David Eigen, Christian Puhrsch, and Rob Fergus. Depth map prediction from a single image using a multi-scale deep network. *CoRR*, abs/1406.2283, 2014.

[7] Ravi Garg, Vijay Kumar BG, Gustavo Carneiro, and Ian Reid. Unsupervised cnn for single view depth estimation: Geometry to the rescue, 2016.

[8] Iro Laina, Christian Rupprecht, Vasileios Belagiannis, Federico Tombari, and Nassir Navab. Deeper depth prediction with fully convolutional residual networks, 2016.

[9] Anirban Roy and Sinisa Todorovic. Monocular depth estimation using neural regression forest. In *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 5506–5514, 2016.

[10] Jinsun Park, Kyungdon Joo, Zhe Hu, Chi-Kuei Liu, and In So Kweon. Non-local spatial propagation network for depth completion, 2020.

[11] Ang Li, Zejian Yuan, Yonggen Ling, Wanchao Chi, shenghao zhang, and Chong Zhang. A multi-scale guided cascade hourglass network for depth completion. In *Proceedings of the IEEE/CVF Winter Conference on Applications of Computer Vision (WACV)*, March 2020.

[12] Zhiqiang Yan, Kun Wang, Xiang Li, Zhenyu Zhang, Baobei Xu, Jun Li, and Jian Yang. Rignet: Repetitive image guided network for depth completion, 2021.

[13] Wouter Van Gansbeke, Davy Neven, Bert De Brabandere, and Luc Van Gool. Sparse and noisy lidar completion with rgb guidance and uncertainty, 2019.

[14] Vitor Guizilini, Rares Ambrus, Wolfram Burgard, and Adrien Gaidon. Sparse auxiliary networks for unified monocular depth prediction and completion, 2021.

[15] Chao Qu, Ty Nguyen, and Camillo J. Taylor. Depth completion via deep basis fitting, 2019.

[16] Yun Chen, Bin Yang, Ming Liang, and Raquel Urtasun. Learning joint 2d-3d representations for depth completion, 2020.

[17] Shanshan Zhao, Mingming Gong, Huan Fu, and Dacheng Tao. Adaptive context-aware multi-modal network for depth completion. *IEEE Transactions on Image Processing*, 30:5264–5276, 2021.

[18] Mu Hu, Shuling Wang, Bin Li, Shiyu Ning, Li Fan, and Xiaojin Gong. Penet: Towards precise and efficient image guided depth completion, 2021.

[19] Yinpeng Chen, Xiyang Dai, Mengchen Liu, Dongdong Chen, Lu Yuan, and Zicheng Liu. Dynamic convolution: Attention over convolution kernels, 2019.

[20] Michael Bloesch, Jan Czarnowski, Ronald Clark, Stefan Leutenegger, and Andrew J. Davison. Codeslam - learning a compact, optimisable representation for dense visual slam, 2018.

[21] Hidenobu Matsuki, Raluca Scona, Jan Czarnowski, and Andrew J. Davison. Codemapping: Real-time dense mapping for sparse slam using compact scene representations, 2021.

[22] Xingxing Zuo, Nathaniel Merrill, Wei Li, Yong Liu, Marc Pollefeys, and Guoquan Huang. Codevio: Visual-inertial odometry with learned optimizable dense depth, 2020.

[23] Carlos Campos, Richard Elvira, Juan J. Gomez, José M. M. Montiel, and Juan D. Tardós. ORB-SLAM3: An accurate open-source library for visual, visual-inertial and multi-map SLAM. *IEEE Transactions on Robotics*, 37(6):1874–1890, 2021.

[24] J Czarnowski, T Laidlow, R Clark, and AJ Davison. Deepfactors: Real-time probabilistic dense monocular slam. *IEEE Robotics and Automation Letters*, 5:721–728, 2020.

[25] Byeong-Uk Lee, Kyunghyun Lee, and In So Kweon. Depth completion using plane-residual representation, 2021.

[26] Saif Imran, Xiaoming Liu, and Daniel Morris. Depth completion with twin surface extrapolation at occlusion boundaries, 2021.

[27] Yinda Zhang and Thomas A. Funkhouser. Deep depth completion of a single RGB-D image. *CoRR*, abs/1803.09326, 2018.

[28] Xiaojuan Qi, Renjie Liao, Zhengzhe Liu, Raquel Urtasun, and Jiaya Jia. Geonet: Geometric neural network for joint depth and surface normal estimation. In *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 283–291, 2018.

[29] Xiaojuan Qi, Zhengzhe Liu, Renjie Liao, Philip H. S. Torr, Raquel Urtasun, and Jiaya Jia. GeoNet++: Iterative geometric neural network with edge-aware refinement for joint depth and surface normal estimation. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 44(2):969–984, feb 2022.

[30] Yan Xu, Xinge Zhu, Jianping Shi, Guofeng Zhang, Hujun Bao, and Hongsheng Li. Depth completion from sparse lidar data with depth-normal constraints, 2019.

[31] Jiaxiong Qiu, Zhaopeng Cui, Yinda Zhang, Xingdi Zhang, Shuaicheng Liu, Bing Zeng, and Marc Pollefeys. Deeplidar: Deep surface normal guided depth prediction for outdoor scene from sparse lidar data and single color image. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2019.

[32] Clément Godard, Oisin Mac Aodha, Michael Firman, and Gabriel Brostow. Digging into self-supervised monocular depth estimation, 2018.

[33] Tinghui Zhou, Matthew Brown, Noah Snavely, and David G. Lowe. Unsupervised learning of depth and ego-motion from video, 2017.

[34] Zhenheng Yang, Peng Wang, Wei Xu, Liang Zhao, and Ramakant Nevatia. Unsupervised learning of geometry with edge-aware depth-normal consistency, 2017.

[35] Fangchang Ma, Guilherme Venturelli Cavalheiro, and Sertac Karaman. Self-supervised sparse-to-dense: Self-supervised depth completion from lidar and monocular camera, 2018.

[36] Fangchang Ma and Sertac Karaman. Sparse-to-dense: Depth prediction from sparse depth samples and a single image. 2018.

[37] Shreyas S Shivakumar, Ty Nguyen, Ian D. Miller, Steven W. Chen, Vijay Kumar, and Camillo J Taylor. Dfusenet: Deep fusion of rgb and sparse depth information for image guided dense depth completion. *https://arxiv.org/pdf/1902.00761.pdf*, 2019.

[38] Yanchao Yang, Alex Wong, and Stefano Soatto. Dense depth posterior (ddp) from single image and sparse range, 2019.

[39] Shing Yan Loo, Ali Jahani Amiri, Syamsiah Mashohor, Sai Hong Tang, and Hong Zhang. Cnn-svo: Improving the mapping in semi-direct visual odometry using single-image depth prediction, 2018.

[40] Christian Forster, Matia Pizzoli, and Davide Scaramuzza. SVO: Fast semi-direct monocular visual odometry. In *IEEE International Conference on Robotics and Automation (ICRA)*, 2014.

[41] Xiaochuan Yin, Xiangwei Wang, Xiaoguo Du, and Qijun Chen. Scale recovery for monocular visual odometry using depth estimated with deep convolutional neural fields. In *2017 IEEE International Conference on Computer Vision (ICCV)*, pages 5871–5879, 2017.

[42] Nan Yang, Rui Wang, Jörg Stückler, and Daniel Cremers. Deep virtual stereo odometry: Leveraging deep depth prediction for monocular direct sparse odometry, 2018.

[43] Nathaniel Merrill, Patrick Geneva, and Guoquan Huang. Robust monocular visual-inertial depth completion for embedded systems. In *IEEE International Conference on Robotics and Automation (ICRA)*, pages 5713–5719. IEEE, 2021.

[44] Patrick Geneva, Kevin Eckenhoff, Woosik Lee, Yulin Yang, and Guoquan Huang. OpenVINS: a research platform for visual-inertial estimation. In *IEEE International Conference on Robotics and Automation (ICRA)*, pages 4666–4672. IEEE, 2020.

[45] Kourosh Sartipi, Tien Do, Tong Ke, Khiem Vuong, and Stergios I. Roumeliotis. Deep depth estimation from visual-inertial slam, 2020.

[46] Luyang Zhu, Arsalan Mousavian, Yu Xiang, Hammad Mazhar, Jozef van Eenbergen, Shoubhik Debnath, and Dieter Fox. Rgb-d local implicit function for depth completion of transparent objects. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2021.

[47] Jonas Uhrig, Nick Schneider, Lukas Schneider, Uwe Franke, Thomas Brox, and Andreas Geiger. Sparsity invariant cnns. In *International Conference on 3D Vision (3DV)*, 2017.

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

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

[50] Angel Chang, Angela Dai, Thomas Funkhouser, Maciej Halber, Matthias Niessner, Manolis Savva, Shuran Song, Andy Zeng, and Yinda Zhang. Matterport3d: Learning from rgb-d data in indoor environments. *International Conference on 3D Vision (3DV)*, 2017.

[51] Richard Hartley and Andrew Zisserman. *Multiple View Geometry in Computer Vision*. Cambridge University Press, USA, 2 edition, 2003.

[52] Tomas Pajdla. *Elements of Geometry for Computer Vision*. 2017.

[53] Vincent Dumoulin and Francesco Visin. A guide to convolution arithmetic for deep learning, 2016.

[54] Stephen Boyd and Lieven Vandenberghe. *Convex optimization*. Cambridge university press, 2004.

[55] Boxin Zhao, Tianjiang Hu, and Lincheng Shen. Visual odometry - a review of approaches. In *2015 IEEE International Conference on Information and Automation*, pages 2569–2573, 2015.

[56] Hugh Durrant-Whyte and Tim Bailey. Simultaneous localisation and mapping (slam): Part i the essential algorithms. *Robotics and Automation Magazine*, 13, 01 2006.

[57] Tim Bailey and Hugh Durrant-Whyte. Simultaneous localization and mapping (slam): Part ii. *Robotics and Automation Magazine, IEEE*, 13:108 – 117, 10 2006.

[58] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition, 2015.

[59] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition, 2014.

[60] Yue Luo, Jimmy Ren, Mude Lin, Jiahao Pang, Wenxiu Sun, Hongsheng Li, and Liang Lin. Single view stereo matching, 2018.

[61] Eduardo Romera, José M. Álvarez, Luis M. Bergasa, and Roberto Arroyo. Erfnet: Efficient residual factorized convnet for real-time semantic segmentation. *IEEE Transactions on Intelligent Transportation Systems*, 19(1):263–272, 2018.

[62] 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.

[63] Peter J. Huber. Robust Estimation of a Location Parameter. *The Annals of Mathematical Statistics*, 35(1):73 – 101, 1964.

[64] Laurent Zwald and Sophie Lambert-Lacroix. The berhu penalty and the grouped effect, 2012.

[65] Ethan Rublee, Vincent Rabaud, Kurt Konolige, and Gary Bradski. Orb: An efficient alternative to sift or surf. In *2011 International Conference on Computer Vision*, pages 2564–2571, 2011.

[66] Edward Rosten and Tom Drummond. Machine learning for high-speed corner detection. In Aleš Leonardis, Horst Bischof, and Axel Pinz, editors, *Computer Vision – ECCV 2006*, pages 430–443, Berlin, Heidelberg, 2006. Springer Berlin Heidelberg.

[67] Michael Calonder, Vincent Lepetit, Christoph Strecha, and Pascal Fua. Brief: Binary robust independent elementary features. In Kostas Daniilidis, Petros Maragos, and Nikos Paragios, editors, *Computer Vision – ECCV 2010*, pages 778–792, Berlin, Heidelberg, 2010. Springer Berlin Heidelberg.

[68] Berthold Horn, Hugh Hilden, and Shahriar Negahdaripour. Closed-form solution of absolute orientation using orthonormal matrices. *Journal of the Optical Society of America A*, 5:1127–1135, 07 1988.

[69] Mathieu Labbé and François Michaud. Appearance-based loop closure detection for online large-scale and long-term operation. *IEEE Transactions on Robotics*, 29(3):734–745, 2013.

[70] Mathieu Labbé and François Michaud. Online global loop closure detection for large-scale multi-session graph-based slam. In *2014 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 2661–2666, 2014.

[71] Mikael Persson and Klas Nordberg. Lambda twist: An accurate fast robust perspective three point (p3p) solver. In *Proceedings of the European Conference on Computer Vision (ECCV)*, September 2018.

[72] Vincent Lepetit, Francesc Moreno-Noguer, and Pascal Fua. Epnp: An accurate o(n) solution to the pnp problem. *International Journal of Computer Vision*, 81, 02 2009.

[73] 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*, 24:381–395, 1981.

[74] Anna Bosch, Andrew Zisserman, and Xavier Munoz. Image classification using random forests and ferns. In *2007 IEEE 11th International Conference on Computer Vision*, pages 1–8, 2007.

[75] Kevin Q. Brown. Voronoi diagrams from convex hulls. *Information Processing Letters*, 9(5):223–228, 1979.

[76] Hauke Strasdat, Andrew J. Davison, J.M.M. Montiel, and Kurt Konolige. Double window optimisation for constant time visual slam. In *2011 International Conference on Computer Vision*, pages 2352–2359, 2011.

[77] Ruo Zhang, Ping-Sing Tsai, J.E. Cryer, and M. Shah. Shape-from-shading: a survey. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 21(8):690–706, 1999.

[78] Supasorn Suwajanakorn, Carlos Hernandez, and Steven M. Seitz. Depth from focus with your mobile phone. In *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 3497–3506, 2015.

[79] Daniel Herrera C., Juho Kannala, L'ubor Ladický, and Janne Heikkilä. Depth map inpainting under a second-order smoothness prior. In Joni-Kristian Kämäräinen and Markus Koskela, editors, *Image Analysis*, pages 555–566, Berlin, Heidelberg, 2013. Springer Berlin Heidelberg.

[80] Junyi Liu and Xiaojin Gong. Guided depth enhancement via anisotropic diffusion. In Benoit Huet, Chong-Wah Ngo, Jinhui Tang, Zhi-Hua Zhou, Alexander G. Hauptmann, and Shuicheng Yan, editors, *Advances in Multimedia Information Processing – PCM 2013*, pages 408–417, Cham, 2013. Springer International Publishing.

[81] Alexey Dosovitskiy, German Ros, Felipe Codevilla, Antonio Lopez, and Vladlen Koltun. CARLA: An open urban driving simulator. In *Proceedings of the 1st Annual Conference on Robot Learning*, pages 1–16, 2017.

[82] Dorian Gálvez-López and J. D. Tardós. Bags of binary words for fast place recognition in image sequences. *IEEE Transactions on Robotics*, 28(5):1188–1197, October 2012.

[83] Nathanael L. Baisa. Derivation of a constant velocity motion model for visual tracking, 2020.

[84] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization, 2014.

[85] Stanford Artificial Intelligence Laboratory et al. Robotic operating system.

[86] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Kopf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. Pytorch: An imperative style, high-performance deep learning library. In H. Wallach,

H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, editors, *Advances in Neural Information Processing Systems 32*, pages 8024–8035. Curran Associates, Inc., 2019.

[87] Matteo Poggi, Filippo Aleotti, Fabio Tosi, and Stefano Mattoccia. On the uncertainty of self-supervised monocular depth estimation, 2020.

[88] Xinyu Nie, Dianxi Shi, Ruihao Li, Zhe Liu, and Xucan Chen. Uncertainty-aware self-improving framework for depth estimation. *IEEE Robotics and Automation Letters*, 7(1):41–48, 2022.

[89] Dongmin Park, Seokil Hong, Bohyung Han, and Kyoung Mu Lee. Continual learning by asymmetric loss approximation with single-side overestimation, 2019.

[90] Z. Chen, B. Liu, R. Brachman, P. Stone, and F. Rossi. *Lifelong Machine Learning: Second Edition*. Synthesis Lectures on Artificial Intelligence and Machine Learning. Morgan & Claypool Publishers, 2018.

Appendix A
Pinhole Camera Model

Since the pinhole model is most commonly used, to model the projection of a 3D point to a 2D Image plane, and is central to a number of works, including concepts described in this work, it is worth introducing it.

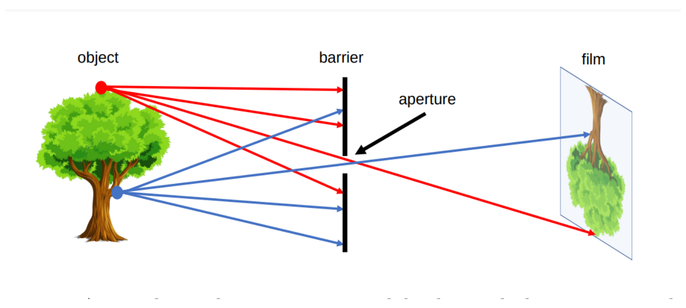


Figure A.1: Simple Pinhole model

As figure 2.2 shows, each point on the object emits multiple rays towards the image plane (film). However, if we place a barrier of low aperture, or a 'PinHole', we get a 1-to-1 relation between a 3D point in the object, and a 2D point in the image plane.

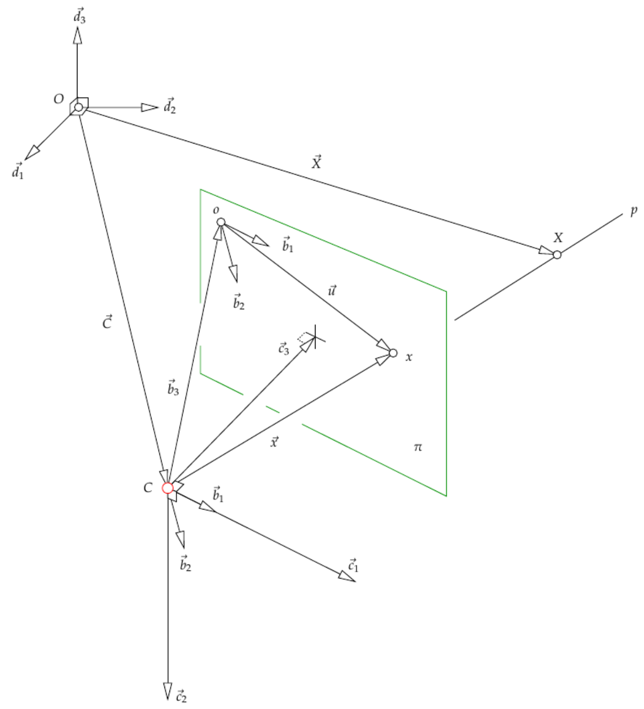


Figure A.2: Coordinate Systems for perspective model. Image from [52]

Consider a 3D point X in some world coordinate frame δ, with the coordinates $X_\delta = [x, y, z]^T$; C is the camera center, and π is the image plane. The pixel coordinates of the projection of this point, x, in the image plane are given by a $[u, v]^T$ in coordinate system α with origin o, and bases $[\vec{b1}, \vec{b2}]$. The coordinate system β is constructed with its origin at C, and bases as $[b1, b2, b3]$ as shown in A.2 where $b1$ and $b2$ are the same as in system α. Thus, the coordinates of the point x in system β is given by:

$$x_\beta = [u, v, 1]^T \tag{A.1}$$

The point X_δ is transformed from the World coordinate system δ to the point X_γ in the camera coordinate system γ with origin C, and bases $[\vec{c1}, \vec{c2}, \vec{c3}]$, by an affine transormation $\in SE(3)$; a Rotation $R \in SO(3)$, and a translation t, a 3×1 vector i.e..

$$X_\gamma = RX_\delta + t \tag{A.2}$$

, or

$$\begin{bmatrix} X_\gamma \\ 1 \end{bmatrix} = \begin{bmatrix} R & t \\ 0^T & 1 \end{bmatrix} \begin{bmatrix} X_\delta \\ 1 \end{bmatrix} \tag{A.3}$$

the 4×4 affine transformation matrix is often called the extrinsics matrix, and the 4D points are points in homogeneous coordinates [51].

This point X_γ is then transformed to the coordinate system β with origin C and bases $[\vec{b1}, \vec{b2}, \vec{b3}]$ by the camera intrinsic matrix denoted by K, which is a 3x3 matrix of the form:

$$K = \begin{bmatrix} k11 & k12 & p_x \\ 0 & k22 & p_y \\ 0 & 0 & 1 \end{bmatrix} \tag{A.4}$$

, which, for rectangular pixels, is of the form

$$\begin{bmatrix} k11 & 0 & p_x \\ 0 & k22 & p_y \\ 0 & 0 & 1 \end{bmatrix} \tag{A.5}$$

where $k11 = k22$ for square pixels.
The parameters $k11, k22$ are ratios pixels/metric units which are called focal lengths f_x, and f_y along the x, and y directions; and p_x and p_y are the pixel coordintaes of the 'principal point'. This matrix K transforms coordinates from the coordinate system γ(C, $[\vec{c1}, \vec{c2}, \vec{c3}]$) to the system β(C, $[\vec{b1}, \vec{b2}, \vec{b3}]$), i.e.

$$X_\beta = \begin{bmatrix} x_\beta \\ y_\beta \\ z_\beta \end{bmatrix} = K \begin{bmatrix} x_\gamma \\ y_\gamma \\ z_\gamma \end{bmatrix} \tag{A.6}$$

We see that the 3D point X, and its projection x, are related in the following way, in system β

$$X_\beta = \lambda \vec{x_\beta} \tag{A.7}$$

where λ in this case is z_β.
Or

$$\begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = \begin{bmatrix} x_\beta/z_\beta \\ y_\beta/z_\beta \\ 1 \end{bmatrix} \tag{A.8}$$

which gives us pixel coordinates

$$\begin{bmatrix} u \\ v \end{bmatrix} = \begin{bmatrix} x_\beta/z_\beta \\ y_\beta/z_\beta \end{bmatrix} \tag{A.9}$$

It is also obvious now that for any value of $\lambda \in \mathbb{R}$, i.e. for any point along the direction vector given by x, the pixel coordinates remain the same.
Putting it all together, we then get,

$$\lambda \begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = P \begin{bmatrix} X_\delta \\ 1 \end{bmatrix} \tag{A.10}$$

where

$$P = \begin{bmatrix} K & \vec{0} \end{bmatrix} \begin{bmatrix} R & t \\ 0^T & 1 \end{bmatrix} \tag{A.11}$$

is a 3×4 projection matrix.
In most real world systems, there are various degrees, and types of distortion present as well, which have to be included in the model.

Appendix B

Network operations

B.1 Pooling

The common pooling functions used are

- Average pooling
- Max/Min pooling

Out of these, the most relevant to this work are the max, and min pooling which are described here:

Max Pooling. Max pooling selects the maximum element from each region of some kernel/filter size. Thus, the output of a max pooling would be a feature map containing the most prominent features of the previous layer.

Consider the following 4x4 block: $\begin{bmatrix} 2 & 2 & 7 & 3 \\ 9 & 4 & 6 & 1 \\ 8 & 5 & 2 & 4 \\ 3 & 1 & 2 & 6 \end{bmatrix}$; with a 2x2 kernel, we operate first on the sub-block $max(\begin{bmatrix} 2 & 2 \\ 9 & 4 \end{bmatrix}) = 9$, and then $max(\begin{bmatrix} 7 & 3 \\ 6 & 1 \end{bmatrix}) = 7$, and so on.

In this case the result of our max pooling with a 2x2 kernel, where we move by a (2,2) stride, and without any additional padding is:

$$\begin{bmatrix} 9 & 7 \\ 8 & 6 \end{bmatrix}$$

Padding can be seen as 'adding' values outside the border to anchor the kernel at, or near the border points, depending on required input, and output sizes of the filter maps. Max pooling is often done with a padding of $-\infty$, a padding of size 1 on our block would look like the following:

$$\begin{bmatrix} -\infty & -\infty & -\infty & -\infty & -\infty & -\infty \\ -\infty & 2 & 2 & 7 & 3 & -\infty \\ -\infty & 9 & 4 & 6 & 1 & -\infty \\ -\infty & 8 & 5 & 2 & 4 & -\infty \\ -\infty & 3 & 1 & 2 & 6 & -\infty \\ -\infty & -\infty & -\infty & -\infty & -\infty & -\infty \end{bmatrix}$$

and applying the same 2x2 kernel max pooling with a (2,2) on this block would give a 3x3 output: $\begin{bmatrix} 2 & 7 & 3 \\ 9 & 6 & 4 \\ 3 & 2 & 6 \end{bmatrix}$ The size of the output can be determined with

$$output\_size = (\frac{n_h - k_h + 2 * p_h}{s_h} + 1) \times (\frac{n_w - k_w + 2 * p_w}{s_w} + 1) \quad (B.1)$$

where $n_h \times n_w$ is the size (rows x cols) of the matrix, $k_h \times k_w$ is the size of the kernel, (s_h, s_w) is the stride along height, and width, and p_h, p_w is the padding size along the height, and width.

Min, Average Pooling. Min Pooling is done similarly by taking the minimum element in each sub-block instead, and is usally padded with ∞ values, and average pooling takes the average of the sub-block, usually with 0 padding values.

B.2 Convolution

As the name implies, convolutions are vital operations in Convolutional Neural Networks (CNNs). A convolution layer applies a convolution filter with a kernel or a filter which approximates to a linear function.

Consider the following 3x3 matrix x:

$$\begin{bmatrix} x_{11} & x_{12} & x_{13} \\ x_{21} & x_{22} & x_{23} \\ x_{31} & x_{32} & x_{33} \end{bmatrix}$$

and the 2x2 convolution kernel w: $\begin{bmatrix} w_{11} & w_{12} \\ w_{21} & w_{22} \end{bmatrix}$ The convolution operation with no padding, and single stride would then give us the output y as in fig B.1 :

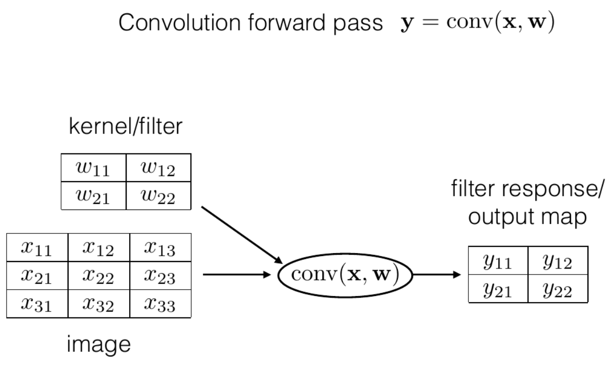

Convolution forward pass $\mathbf{y} = \text{conv}(\mathbf{x}, \mathbf{w})$
kernel/filter
w_{11} w_{12} w_{21} w_{22}
x_{11} x_{12} x_{13} x_{21} x_{22} x_{23} x_{31} x_{32} x_{33}
image
$\text{conv}(\mathbf{x}, \mathbf{w})$
filter response/ output map
y_{11} y_{12} y_{21} y_{22}

Figure B.1: Convolution operation

where,

$$y_{11} = w_{11}x_{11} + w_{12}x_{12} + w_{21}x_{21} + w_{22}x_{22}$$
$$y_{12} = w_{11}x_{12} + w_{12}x_{13} + w_{21}x_{22} + w_{22}x_{23}$$
$$y_{21} = w_{11}x_{21} + w_{12}x_{22} + w_{21}x_{31} + w_{22}x_{32}$$
$$y_{22} = w_{11}x_{22} + w_{12}x_{23} + w_{21}x_{32} + w_{22}x_{33}$$

The padding, stride, and their relation to the output size work can similarly obtained from equation (B.1). However, the padding values for convolution are usually 0,1,etc..

B.3 Upconvolution and Transpose Convolutions

In [1] , UpConvolution or transpose convolutions are the two methods the decoder uses in its layers. The UpConvolution consists of an upsampling + a conventional convolution.

Upconvolution. first interpolates the input with a nearest neighbor interpolation to the required output shape, and then performs a convolution on the output of the interpolation to upsample a feature map.

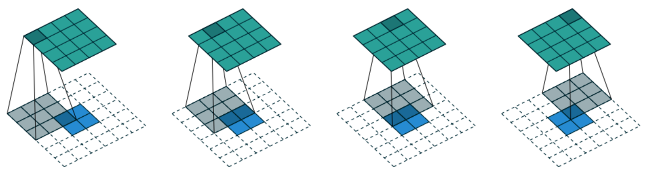
Transposed Convolution. also called fractionally strided convolution, and (incorrectly) deconvolution work by swapping the forward and backward passes of a convolution. The trasposed convolution can be considered as an operation that allows to recover the shape of the initial feature map.

Consider the convolution of a 3×3 kernel on a 4×4 input with single stride and no padding. Using equation (B.1) , we see that this produces a 2×2 output. The transpose of this convolution will then have an output of shape 4×4 when applied on a 2×2 input.

Let the input size of the original block (4x4 in our example) be $n_h^i \times n_w^i$; the output size of the original convolution (2x2 in our example) be $n_h^o \times n_w^o$, with kernel size $k_h \times k_h$ with stride (s_h, s_w). We consider the case where there is no padding for the original convolution.
One way to achieve the transpose convolution is by convolving a 3×3 kernel over the 2×2 input (output of the original convolution) padded with a 2×2 padding using single (1,1) stride. We can again verify with equation (B.1) that this operation now gives us a 4×4 feature map, which is the same size as the original input. The kernel, and stride have the same size as the original, but the input is now padded. The required padding size for a single stride can easily be calculated with the following:

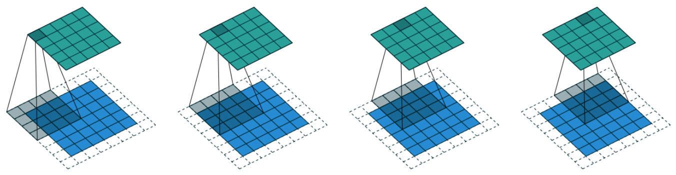
$$p^t = \frac{n_h^i - n_h^o + k_h - 1}{2}, \frac{n_w^i - n_w^o + k_w - 1}{2}) \tag{B.2}$$

where p^t is the padding size required for the transpose convolution. Using this on our example, we see that p^t is indeed $(2, 2)$



The transpose of convolving a 3×3 kernel over a 4×4 input using unit strides (i.e., $i = 4$, $k = 3$, $s = 1$ and $p = 0$). It is equivalent to convolving a 3×3 kernel over a 2×2 input padded with a 2×2 border of zeros using unit strides (i.e., $i' = 2$, $k' = k$, $s' = 1$ and $p' = 2$).

Figure B.2: Transpose Convolution operation eg without padding in the original convolution. Image from [53]



The transpose of convolving a 4×4 kernel over a 5×5 input padded with a 2×2 border of zeros using unit strides (i.e., $i = 5$, $k = 4$, $s = 1$ and $p = 2$). It is equivalent to convolving a 4×4 kernel over a 6×6 input padded with a 1×1 border of zeros using unit strides (i.e., $i' = 6$, $k' = k$, $s' = 1$ and $p' = 1$).

Figure B.3: Transpose Convolution operation eg with padding in the original convolution. Image from [53]

Appendix C

Some properties of affine rigid motion transformations

The special Euclidean group in 3 dimension, called SE(3), contains the set of rigid transformations with some rotation R, and translation t. A member $p \in SE(3)$ is of the form:

$$p = \begin{bmatrix} R & t \\ \vec{0} & 1 \end{bmatrix} \tag{C.1}$$

where $R \in SO(3)$ is the rotation matrix, t is the 3×1 translation vector $\in \mathbb{R}^3$, $\vec{0}$ is a 1×3 vector of all 0s.

This transformation can be seen in two ways;

- a transformation of a point X in some coordinate frame α to another point Y in the same coordinate frame; i.e.

$$\begin{bmatrix} Y_\alpha \\ 1 \end{bmatrix} = p \begin{bmatrix} X_\alpha \\ 1 \end{bmatrix} \tag{C.2}$$

or

- a change of basis a point X from some coordinate frame β to coordinate frame α; i.e.

$$\begin{bmatrix} X_\alpha \\ 1 \end{bmatrix} = p \begin{bmatrix} X_\beta \\ 1 \end{bmatrix} \tag{C.3}$$

  where, in this case, the columns of R contain the coordinates of the basis vectors of the frame β in the frame α, and the translation t is the coordinates of the origin of frame β in the frame α

It is important to see two important properties of this group:

Composition. for any $p, q \in SE(3)$ their composition, which is a matrix product

$$pq \in SE(3) \tag{C.4}$$

and this pq means that we first apply the transformation q, and then apply p.

Inverse. The inverse of the matrix p gives us the inverse transformation, i.e. if

$$\begin{bmatrix} X_\alpha \\ 1 \end{bmatrix} = p \begin{bmatrix} X_\beta \\ 1 \end{bmatrix} \tag{C.5}$$

then

$$\begin{bmatrix} X_\beta \\ 1 \end{bmatrix} = p^{-1} \begin{bmatrix} X_\alpha \\ 1 \end{bmatrix} \tag{C.6}$$

For showing (C.4): Let $p = \begin{bmatrix} R_1 & t_1 \\ \vec{0} & 1 \end{bmatrix}$ and $q = \begin{bmatrix} R_2 & t_2 \\ \vec{0} & 1 \end{bmatrix}$

$$pq = \begin{bmatrix} R_1 & t_1 \\ \vec{0} & 1 \end{bmatrix} \begin{bmatrix} R_2 & t_2 \\ \vec{0} & 1 \end{bmatrix} = \begin{bmatrix} R_1R_2 + t_1\vec{0} & R_1t_2 + t_1 \\ \vec{0}R_2 + 1.\vec{0} & \vec{0}.t2 + 1 \end{bmatrix} = \begin{bmatrix} R_1R_2 & R_1t_2 + t_1 \\ \vec{0} & 1 \end{bmatrix} \tag{C.7}$$

the product of the two rotations R_1R_2 is another rotation matrix; this is geometrically obvious that one rotation after another, can be represented as another rotation, and $R_1t_2 + t_1$ is a 3×1 translation vector.

For showing (C.6): Let $p = \begin{bmatrix} R & t \\ \vec{0} & 1 \end{bmatrix}$ and $p^{-1} = \begin{bmatrix} Q & \vec{y} \\ \vec{0} & 1 \end{bmatrix}$ where Q is some 3×3 matrix, and $\vec{y}$ is some translation: The inverse should sastisfy:

$$pp^{-1} = \begin{bmatrix} I_{3\times3} & \vec{0^T} \\ \vec{0} & 1 \end{bmatrix} \tag{C.8}$$

which corresponds to no rotation, and 0 translation.

From this, we get that

$$RQ + t.\vec{0} = I_{3\times3} \tag{C.9}$$

which implies

$$Q = R^{-1} = R^T \tag{C.10}$$

and

$$R\vec{y} + t = \vec{0^T} \tag{C.11}$$

which implies

$$\vec{y} = -R^{-1}t = -R^Tt \tag{C.12}$$

which then gives

$$p^{-1} = \begin{bmatrix} R^T & -R^Tt \\ \vec{0} & 1 \end{bmatrix} \tag{C.13}$$

Now let us look at the change of basis from β to α with:

$$X_\alpha = RX_\beta + t \tag{C.14}$$

With some basic algebraic manipulation:

$$RX_\beta = X_\alpha - t \implies X_\beta = R^TX_\alpha - R^Tt \tag{C.15}$$

which can be shown with an affine transformation as:

$$\begin{bmatrix} X_\beta \\ 1 \end{bmatrix} = \begin{bmatrix} R^T & -R^T t \\ \vec{0} & 1 \end{bmatrix} \begin{bmatrix} X_\alpha \\ 1 \end{bmatrix} \tag{C.16}$$

and from (C.13) is the same as (C.6)

Appendix D

Miscellaneous

D.1 Epipolar Geometry

Epipolar Geometry details geometrical constraints for multi-view scene geometry.

Consider two cameras with camera centers C_1, and C_2 observing a 3D point in the world X as shown in figure D.1 with poses R_1, t_1 and R_2, t_2 respectively with resepct to some world coordinate frame δ, and intrinsic matrices K_1, and K_2. So,

$$\lambda_1 x_1 = P_1 \begin{bmatrix} X_\delta \\ 1 \end{bmatrix} \tag{D.1}$$

and

$$\lambda_2 x_2 = P_2 \begin{bmatrix} X_\delta \\ 1 \end{bmatrix} \tag{D.2}$$

where x_1, x_2 are the homoegeneous coordinates of respective projections of the point X, and P_1, P_2 are the projection matrices of the form (A.11).

The camera center C_1 will have the coordinates $[0, 0, 0]^T$ in the camera 1 frame, and coordinates $C_{1\delta}$. For change of basis (A.2) from world frame δ to camera1 frame γ_1

$$X_{\gamma_1} = R_1 X_\delta + t_1 \tag{D.3}$$

Thus,

$$\vec{0} = R_1 C_{1\delta} + t_1 \tag{D.4}$$

from which we get

$$C_{1\delta} = -R_1^T t_1 \tag{D.5}$$

and similarly

$$C_{2\delta} = -R_2^T t_1 \tag{D.6}$$

Such a scene as in D.1 will satisfy the constraints [PAJDLA TEXT]:

$$x_2^T K_2^{-1} R_2 [C_{2\delta} - C_{1\delta}]_\times R_1^T K_1^{-1} x_1 = 0 \tag{D.7}$$

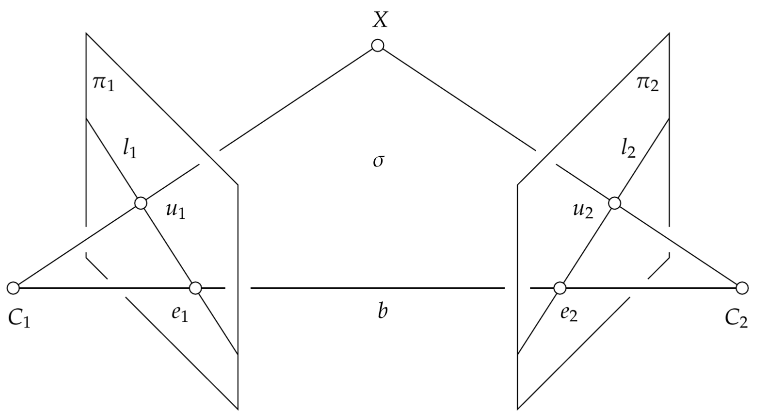


Figure D.1: Epipolar Geometry of two cameras. Image from text [52]

where the $[C_{2\delta} - C_{1\delta}]_\times$ is a skew-symmetric cross product matrix of the vector $\vec{C_{2\delta}} - \vec{C_{1\delta}}$.
This is represented as:

$$x_2^T F x_1 = 0 \tag{D.8}$$

where the Fundamental matrix

$$F = K_2^{-1} R_2 [C_{2\delta} - C_{1\delta}]_\times R_1^T K_1^{-1} \tag{D.9}$$

or

$$x_2^T K_2^{-1} E K_1{-}1 x_1 = 0 \tag{D.10}$$

where the Essential matrix

$$E = R_2 [C_{2\delta} - C_{1\delta}]_\times R_1^T \tag{D.11}$$

Equations (D.8), and (D.10) are known as the epipolar constraints with the Fundamental, and Essential matrix respectively. It can be seen that the Fundamental, and Essential Matrices encodes the relative rotation, and translation between the two cameras C_1 and C_2 which is extremely useful.

D.2 A very brief introduction to some graph concepts

D.2.1 Subgraph

Let $G_1 = (V_1, E_1)$ and $G_2 = (V_2, E_2)$ be two graphs. G_2 is a subgraph of G_1 i.e. $G_2 \subseteq G_1$ if $V_2 \subseteq V_1$ and $E_2 \subseteq \{E_1 \ \textit{restricted to} \ V_2\}$ where V_i are the vertices, or nodes of the graph, and E_i are the edges.
For two graphs G_1, G_2 having the same vertex set, G_2 is a subgraph of G_1 iff its edge set E_2 is a subset of E_1, i.e. $G_2 \subseteq G_1 \ iff E_2 \subseteq E_1$

D.2.2 Trees

A tree is an acyclic connected graph of n vertices, and $n-1$ edges. It implies a hierarchical structure with parent nodes emanating edges which connect to children nodes.
In a tree, there exists a node called root, which has no parents, and this is the highest node in the tree hierarchy.
The nodes which have no children are called leaves, which are the lowest nodes in the hierarchy.

Spanning Tree. is a tree subgraph containing *all* the original graph vertices, i.e. for a graph $G_1(V_1, E_1)$, a **tree** $ST(V_1, E_2) \subseteq G_1$.