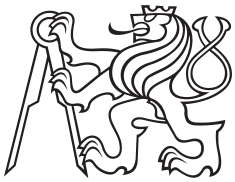
Bachelor Thesis



Czech Technical University in Prague

F3

Faculty of Electrical Engineering
Department of Cybernetics

Machine Learning for Robotic Exploration

Jáchym Staněk

Supervisor: MSc. Ruslan Agishev
Field of study: Cybernetics and robotics
May 2022

Acknowledgements

Foremost, I would like to express my sincere gratitude to my supervisor MSc. Ruslan Agishev for his continuous support. It was his advice and guidance that made this work possible.

I would like to thank the Center for Machine Perception (CMP) for providing me with access to the computational servers.

Lastly, I would like to express my deepest appreciation to my family and friends for supporting me during my work on the thesis.

Declaration

I declare that the presented work was developed independently and that I have listed all sources of information used within it in accordance with the methodical instructions for observing the ethical principles in preparation of university theses.

Prague, 20. May 2022

Abstract

This work, we proposes, implements and evaluates a novel way to train depth completion networks using map reconstruction error. A traditional way to train depth completion networks with MSE was also implemented, and both methods were compared.

We have included a differentiable dense SLAM module in our learning pipeline and evaluated the model on the KITTI dataset. We introduced depth completion network to make input to SLAM more dense in order to provide more correspondences to work with.

Using the depth completion network, we were able to obtain denser depth maps. Training with the map reconstruction error yielded results similar to those of traditional methods. Denser data did not increase SLAM localization accuracy, this was mainly because the model introduced too many outliers and made it difficult for SLAM to work.

The code is publicly available at: `https://github.com/jachym-stanek/supervised_depth_correction`

Keywords: Machine learning, SLAM, Depth completion, KITTI

Supervisor: MSc. Ruslan Agishev

Abstrakt

Tato práce navrhuje, implementuje a vyhodnocuje nový způsob trénování sítí pro doplňování hloubky pomocí chyby rekonstrukce mapy. Byl také implementován tradiční způsob trénování hloubkových doplňovacích sítí pomocí MSE a obě metody byly porovnány.

Do učebního procesu jsme zahrnuli modul diferencovatelného hustého SLAMu a model jsme vyhodnotili na datové sadě KITTI. Implementovali jsme síť pro doplňování hloubky, aby byl vstup do SLAMu hustší a poskytl tak více korespondencí, se kterými lze pracovat.

Pomocí sítě pro doplňování hloubky jsme byli schopni získat hustší hloubkové mapy. Trénování s chybou rekonstrukce mapy přineslo podobné výsledky jako tradiční metody. Hustší data nezvýšila přesnost lokalizace SLAM, a to hlavně proto, že model vytvořil příliš mnoho odlehlých hodnot a ztížil práci pro SLAM.

Klíčová slova: Strojové učení, SLAM, Doplnění hloubky, KITTI

Překlad názvu: Strojové učení pro robotickou exploraci

Contents

Figures

Tables

Chapter 1

Introduction

Fully differentiable solutions for robotics and computer vision problems have been gaining increasing attention lately. This is because they allow for efficient fine-tuning of large and complex pipelines and provide robust solutions. Some examples include an end-to-end differentiable pipeline for autonomous driving [6], pose consistency loss [7], differentiable model predictive control [8], PyTorch3D [9], and several versions of the differentiable Simultaneous Localization and Mapping (SLAM) [10], [11].

The SLAM module is an essential part of robotic exploration pipelines, it constructs map and provides poses from measurements. SLAM uses correspondences between points, and these can be hard to obtain if there is a low number of points to work with. Methods based on sparse data are not very robust and can fail. This is why methods based on dense SLAM are becoming more popular. Some examples of these include Kinect fusion [12], RGB-D SLAM with volumetric fusion[13] or Point fusion [2]. These methods create denser maps and are more robust.

This work tackles the question of whether an improvement in localization accuracy can be achieved by introducing a machine learning model to improve depth data. As dense SLAM methods should work better with a denser input, we introduce a depth completion model that fills in the missing depth measurements. For the purpose of learning the model, the entire pipeline was made differentiable, this was achieved thanks to the aforementioned differentiable SLAM module [11].

We trained the model in a novel way using the map reconstruction error and the mean square error, which is a standard loss for depth completion networks. We have used the KITTI dataset [5] which provides dense and sparse data and also ground truth trajectories, and thus allowed us to employ supervised training.

1.1 Outline

Chapter 2 introduces the basic concepts and theory on which this thesis is built upon. Briefly describes the SLAM problem and some of its variants, as well as the concepts of robotic exploration and depth completion.

Chapter 3 introduces the topic of dense vs. sparse depth inputs to SLAM more in detail and describes the machine learning pipeline that was used.

Chapter 4 describes the setup of the performed experiment, the datasets, and the scientific and engineering methods that were used in the said experiment.

Chapter 5 provides a more in-depth summary of the experimental results and presents numerical results of the evaluation metrics. We present results for both training and testing of the model.

Chapter 6 discusses the results, their application and meaning.

Chapter 7 provides the final summary and concludes the work. Suggestions for future work are also stated.

1.2 Related Work

1.2.1 Convolutional Neural Networks with Local Context Masks

The thesis Convolutional Neural Networks with Local Context Masks [14] proposes a new modified convolution operation with local context mask. The results are evaluated on the KITTI dataset, and one of the investigated uses is depth completion. This work is similar because it also delves into the topic of deep learning for depth completion and uses the KITTI dataset for evaluation. However, the SLAM problem is not a topic there.

1.2.2 Depth completion

Depth completion focuses on the task of creating dense depth maps from sparse measurements by filling in the missing measurements. This area of research has been rapidly developing over the past few years. New methods have been created thanks to the advent of Deep convolutional neural networks (DCNNs).

There are many different approaches to depth completion. Some examples of solutions include: Adaptive Aggregation Network for Efficient Stereo Matching [15] which replaces the commonly used 3D convolutions by a sparse points based intra-scale cost aggregation method, or the Hierarchical Neural Architecture Search for Deep Stereo Matching [16] which proposes end-to-end hierarchical NAS framework for deep stereo matching .

The machine learning model used in this thesis uses the structure proposed in the Sparsity Invariant CNNs paper [4]. This thesis differs in how this model is used and in what goal is aimed to be achieved. Compared to the original paper, we do not only focus on the sole depth completion, but our main goal is the improvement of SLAM performance. The authors of this model tested it on the KITTI dataset [5], which is also used for the purposes of this work.

It might not be immediately clear how the depth completion can be used in real-world tasks, that is why we take this problem a step further. We investigate how we can apply the learned models for SLAM tasks and we also investigate different training methods (losses).

1.2.3 Differentiable SLAM

Differentiable SLAM methods allow machine learning networks to learn robust solutions in challenging conditions because of their ability to optimize all model components jointly for the end objective.

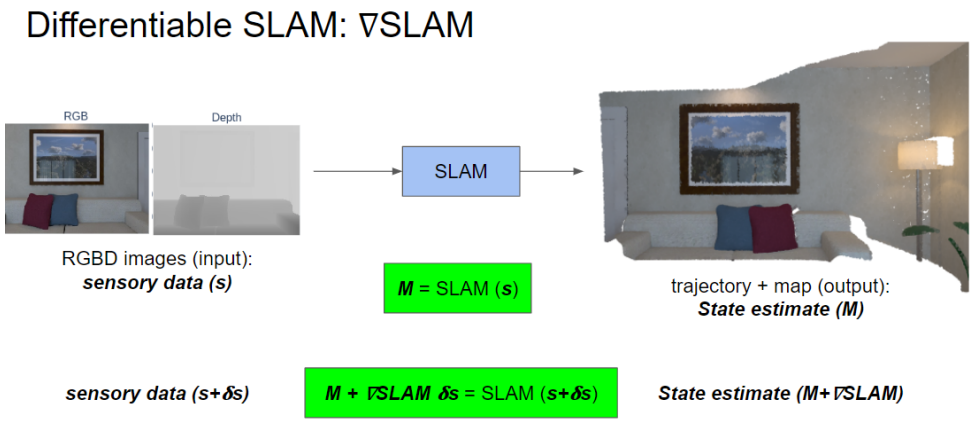


Figure (1.1) : Differentiable SLAM (provided by thesis supervisor)

Differentiable SLAM can be thought of as a function where the input is a sequence of sensory data (RGBD images) and the output is a map and camera trajectory (estimation of state M). Differentiable SLAM allows us to see how small perturbations of input sensory data affect the change of state estimate.

An example of a differentiable SLAM is SLAM-net [10]. This network encodes a particle filter based SLAM algorithm in a differentiable computation graph and learns task-oriented neural network components by backpropagating through the particle filter algorithm.

Another approach is introduced in the gradSLAM: Dense SLAM meets Automatic Differentiation paper [11]. The authors of this paper created separate differentiable versions of non-differentiable SLAM components. GradSLAM

is a module suitable for dense SLAM and represents a crucial component of the learning pipeline constructed for this work. The authors of gradSLAM performed an experiment on the ICL-NUIM dataset [17], this work only used ICL-NUIM for demo and testing and for the final version the KITTI dataset was used.

Chapter 2

Theoretical background

2.1 Robotic Exploration

Robotic exploration is an area of robotics that has been rapidly developing in the last decade. Both hardware and software tools have made great leaps forward in their development, and this in turn enabled more sophisticated and reliable methods to emerge. The most notable advances have been seen in a closely related field of driverless vehicles, with some companies already testing fully autonomous automobiles in live traffic [18]. Autonomous driving closely relates to robotic exploration, as in both cases the agent (be that robot or car) must make their own decisions, plan route's trajectory, scan surrounding terrain, and react accordingly to unpredictable situations (people not following the rules of traffic, slippery surface, etc.). The differences lie in what are the objectives: in robotic exploration, we want to map as much terrain as possible as accurately as possible, while driverless vehicles want to get from point A to point B as efficiently as possible. Robotic exploration can be useful, for example, for exploring environments that are dangerous or inaccessible to humans, such as caves filled with toxic fumes, collapsed buildings, or celestial bodies. An example of this may be the DARPA Subterranean Challenge [19], which focuses on the exploration of underground environments.

Robotic exploration involves one or more robots that explore and map the environment. The goal of robotic exploration can be to explore and map as much of the environment as possible, or it can be, for example, to find points of interest (disaster survivals, toxic waste leaks, etc.). It is preferable that the robot (or robots) performing the exploration task is well suited for the given environment. It can be difficult to explore tight spaces with drones, and a wheeled robot might find it difficult to scale down stairs.

In order to obtain a map of an environment, robots must be equipped with the right tools. Robots are usually equipped with a LiDAR or a depth camera to scan the surrounding environment and find distances of objects around the robot. LiDAR is generally more suitable for outdoor environments, as it suffers less from environmental conditions, such as the weather. On the other hand, the depth camera provides more detailed measurements, making it the

preferred choice for indoor environments. An essential part of any exploration pipeline is SLAM, because it provides robot's location in an environment and allows to build a map.

2.2 SLAM

According to the book "Probabilistic Robotics" [20], Simultaneous localization and mapping (SLAM) is a process in which we construct a 3D map of an unknown environment while simultaneously keeping track of the agent's position in the said environment. This process is sometimes also known as Concurrent Mapping and Localization and poses as one of the most fundamental problems in robotics and a key part of robotic exploration solutions with applications such as navigation of commercial robots. The main problem stems from the fact that the robot performing the exploration does not have access to the map of its environment, nor does it possess knowledge of its pose. Instead, it receives measurements from sensors (for example, camera and lidar) and creates a map from these measurements.

There are two basic versions of the SLAM problem: online and global. Both of these versions involve estimation of the map of the environment, however, they differ in approaches to trajectory estimation. Online SLAM focuses only on the momentary pose of the robot while global SLAM seeks to determine all poses. Both of these SLAM problems are of significant importance, however in this thesis we will focus on the global SLAM and localization accuracy of the trajectory containing all poses.

There are several commonly used ways of representing environment maps, each having its respective positives and negatives. The most basic way of map representation is a 3D pointcloud, which is just a list of points with each point containing x, y and z coordinates. This is a very accurate method, however it is very memory inefficient. A surfel map is a collection of local surface element (surfels). Each surfel is defined by a 3D point and a normal vector. Surfel maps allow for point-to-plane measure (works better on planar surfaces), however they require the surfel updating procedure. Another map representation is the voxel grid. Voxels represent some value in a regular grid. An example of a 2D voxel grid is the division of voxels into 3 categories: occupied, unoccupied, and unknown. This can be a good mapping technique for the exploration of unknown environments.

2.2.1 ICP-SLAM

When constructing a map, we come across the problem of creating correspondences between subsequent pointclouds. These correspondences are needed to create an accurate map and establish which points are new in a measurement and which have already been observed. We also need to find out what the

transformation is between the already constructed map and current measurement to accurately add the new points. Iterative Closest Point (ICP) in one of the commonly used algorithms for matching pointclouds and a useful tool for acquiring point correspondences. Using the ICP algorithm, we can create a basic SLAM algorithm - ICP-SLAM.

This section provides a brief mathematical and algorithmic description of the ICP algorithm based on the description in papers [21] and [22].

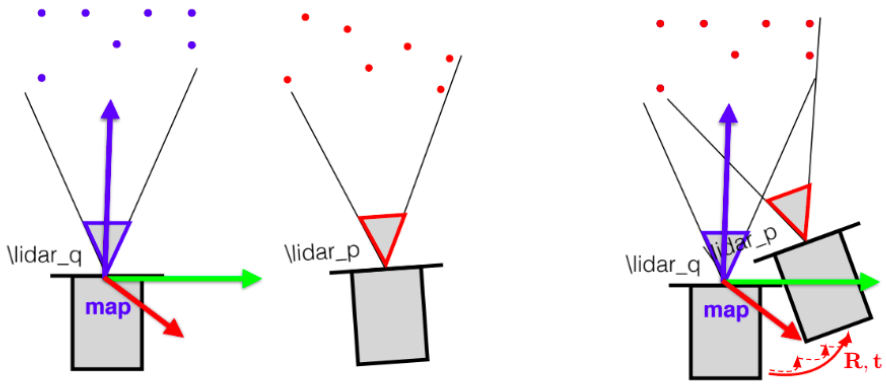


Figure (2.1) : Schematic of the pointcloud alignment in ICP [1]

Let Q be the current map, P the point cloud obtained from the current measurement, the rotational matrix $\boldsymbol{R}$ and the translation vector $\boldsymbol{t}$ be the correction of transformation induced by alignment of P with Q. The ICP algorithm can be described in these steps:

Algorithm 1 ICP

1: Initialize $\boldsymbol{R} = \mathbb{I}$, $\boldsymbol{t} = \boldsymbol{0}$
2: For each point $\boldsymbol{p_i} \in P$ find closest point $\boldsymbol{q_i} \in Q$
3: Reject outliers by median threshold
4: Solve

$$\boldsymbol{R'}, \boldsymbol{t'} = \underset{\boldsymbol{R'} \in SO(3), \boldsymbol{t'} \in \mathbb{R}^3}{\arg\min} \sum \|\boldsymbol{R'p_i} + \boldsymbol{t'} - \boldsymbol{q_i}\|_2^2$$

5: Update P and robot's position:

$$\begin{aligned} \boldsymbol{p_i} &\coloneqq \boldsymbol{R'p_i} + \boldsymbol{t'} \quad \forall i \\ \boldsymbol{R} &\coloneqq \boldsymbol{R'R} \\ \boldsymbol{t} &\coloneqq \boldsymbol{R't} + \boldsymbol{t'} \end{aligned}$$

The ICP algorithm yields $\boldsymbol{R}$ and $\boldsymbol{t}$ from input global map (Q) and current measurement (P).

Once we know how the ICP algorithm works, we can move on to ICP-SLAM: Suppose that Q is the current map, P is the point cloud obtained from the current measurement, $\boldsymbol{R}$ represents the rotation matrix, and $\boldsymbol{t}$ is the translation vector, then ICP-SLAM can be described in these steps:

Algorithm 2 ICP-SLAM

1: Filter P by uniform sampling
2: Align lidar P with map Q by the last known transformation
3: R,t = ICP(P,Q)
4: Update map
5: Update robot's position (R,t)

In this way, we get an algorithm that progressively constructs a global map of the environment from measurements and simultaneously updates the robot's position.

2.2.2 Point Fusion

Point-based fusion (Point fusion for short) is another way of addressing the SLAM problem. We mention it here because it is also implemented in the gradSLAM package [11] and we provide description of this algorithm based on the Point fusion paper [2] from which the implementation in gradSLAM is derived.

In most mapping strategies, the number of map elements increases proportionally to the exploration time. However, this is an undesirable feature. Ideally, the map elements should instead increase proportionally to the volume of explored occupied space. Point fusion mapping techniques employ this feature through the fusion of redundant observations of the same map element into just one point. As a consequence, the recovered map has a more manageable size, and this saves a lot of memory. Another feature is the improvement in the quality of the reconstruction [11].

The point fusion SLAM algorithm has four basic steps: Depth map preprocessing, camera pose estimation, depth map fusion, and dynamics estimation. We can see the pipeline of this algorithm in the picture 2.2.

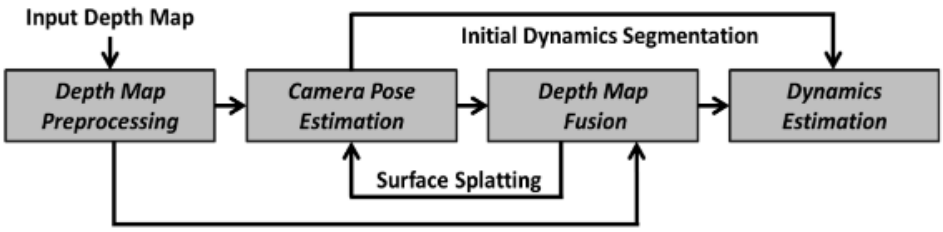


Figure (2.2) : Point fusion pipeline schematic [2]

During the depth map preprocessing, each input depth map is transformed

into a set of 3D points using the intrinsic parameters of the camera and stored in a 2D vertex map.

Depth map fusion takes care of the single global model and fuses input points into it, using a valid camera pose. The global model is simply a list of 3D points with associated attributes (e.g. normals). If the corresponding points are found, the most reliable point is merged with the new point estimate using a weighted average. Otherwise, the new point estimate is added to the global model as an unstable point. The global model is updated over time to remove outliers to improve visibility and alleviate temporal constraints.

Camera pose estimation is done using the ICP algorithm. Dynamics estimation keeps track of dynamic objects and updates their position accordingly.

2.2.3 Dense SLAM

Sparse SLAM suffers from a lack of robustness, as it can be harder to match points with a lack of correspondences. Furthermore, sparse maps can make it difficult to visually identify objects within the map. Opposed to sparse SLAM, dense SLAM methods aim to use the entire camera image as measurements in order to increase robustness and accuracy in estimation. In addition to solving the SLAM problem, 3D reconstructions generated by a dense method are open to a wider range of applications than sparse ones. [23]

The sequence of operations in dense SLAM systems could be summarized as odometry estimation (frame-to-frame alignment), map building (model-to-frame alignment/local optimization), and global optimization. [11]

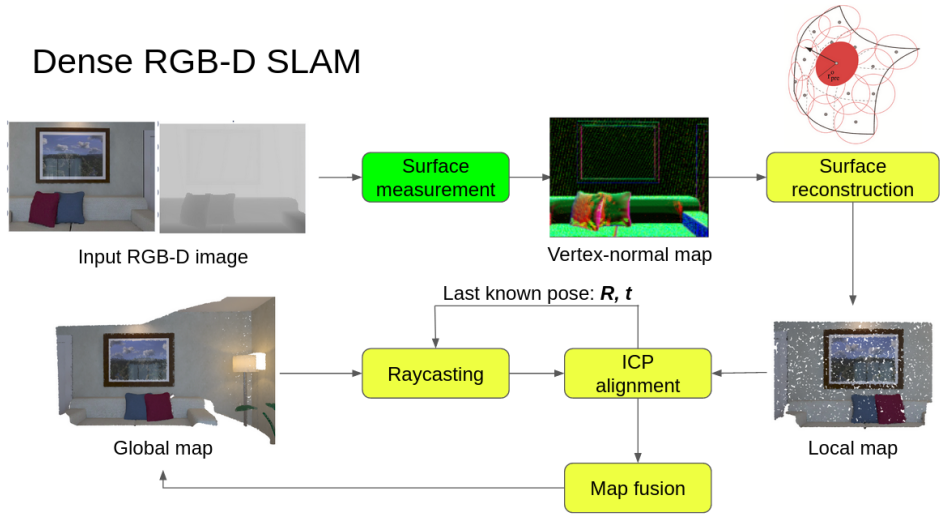


Figure (2.3) : Dense RGBD SLAM schematic (provided by thesis supervisor)

In the picture 2.3 we can see a typical pipeline used in dense SLAM modules. The operations involved are: raycasting, ICP alignment, surface reconstruction, and map fusion. These operations are inherently non-differentiable,

but in the gradSLAM module [11], which we used, these parts were made differentiable.

Sensory measurements are first converted to vertex-normal map, which is then converted into a local map, which is then aligned using the ICP and raycasting with the global map. The aligned map is fused to update the global map. Redundant measurements (measurements of the same point) are removed during the fusion process. This process is repeated for each new coming frame, making it a sequential process, which could be non-differentiable.

Notice the yellow components in the system that are not differentiable. We used a differentiable dense SLAM module ([11]) which makes these blocks differentiable. The technique used for this was based on the idea of computational graphs, and new versions of these components were introduced, allowing them to act as functions.

We are focused on the dense SLAM tasks, as we would like to construct more detailed maps, and also to test the hypothesis, whether dense RGB-D sensor measurements help to improve frame alignment process (ICP).

2.3 Depth Completion

Depth completion is an area of research with a wide variety of applications, mostly for navigation, be that of autonomous robots or self-driving automobiles. The core of depth completion lies in finding a way to obtain dense depth data from sparse depth measurements. This is usually done using a machine learning model. Approaches to training can vary from supervised or semi-supervised to unsupervised losses.

The two most commonly used devices for obtaining depth measurements are LiDAR and depth cameras. LiDARs are often used for mapping outdoor environments, while depth cameras are more used for indoor, close-quarters environments. Data obtained from these measurements might not be perfect, the measuring devices could be imprecise or suffer from unfavorable external conditions - e.g. LiDAR might not work well in foggy tunnels, and cameras might get deceived by direct sunlight. More importantly for depth completion, depth measurements are often sparse, meaning that a significant number of pixels in depth image are lacking values. Depth completion networks aim to reduce this sparsity and create dense depth images by predicting missing values. DCNNs have proven quite good for this task.

Supervised approaches usually use some kind of loss based on the difference between the ground truth depth image and the predicted depth image. One of the most basic metrics used is the MSE. However, it is possible to use different types of loss, in our case we proposed using the map reconstruction error. Some depth completion methods use only the sparse depth measurement (e.g. [4]), but it is also possible to use the RGB image ([24]).

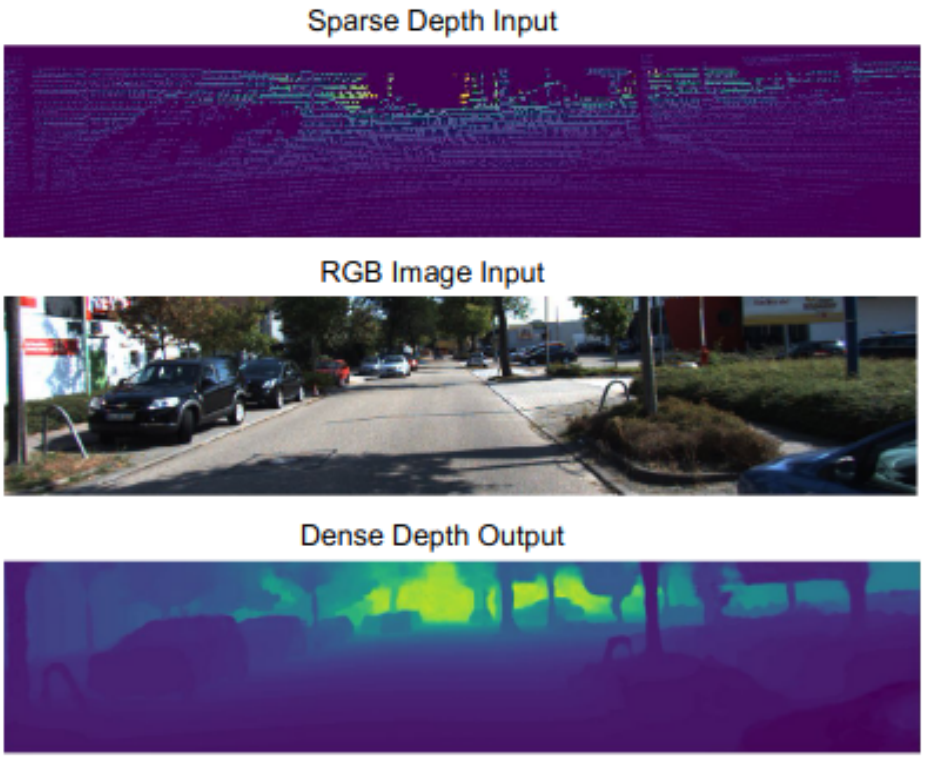


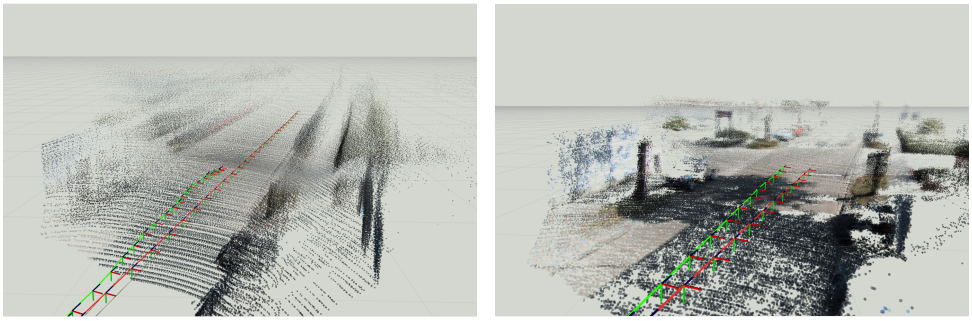
Figure (2.4) : Example of depth completion with sparse depth and RGB image [3]

Chapter 3

Methodology

3.1 RGBD-SLAM with denser input

Depth data acquired from sensors is often imperfect, contains noise, and missing values. These imprecisions can lead to the creation of inaccurate maps and imperfections in localization within the environment. Furthermore, sparse depth images contain a low amount of points which makes it difficult for SLAM to find correct correspondences, this can also cause maps to be less precise.



(a) : Map construction with ICP-SLAM from sparse depth images

(b) : Map construction with ICP-SLAM from dense depth images

Figure (3.1) : Comparison of maps constructed from sparse and dense images with SLAM odometry

Figure C.6b shows the construction of the map in the ROS Rviz simulator using the ICP-SLAM and the KITTI depth dataset. In the two pictures, we can see comparison of the trajectory created by SLAM (green line) and ground truth trajectory (red line). Even the SLAM trajectory from dense images is diverging from ground truth (at the point shown in the picture, the trajectory diverged by 0.9 meters). However, this divergence is relatively small compared to SLAM trajectory created from sparse images. Despite the divergence, the trajectory keeps on par with the ground truth trajectory. On the other hand, the trajectory created from sparse depth images struggles to

keep close to the ground truth trajectory and the poses keep getting more clumped up. This results in a significantly larger error in localization accuracy (at the point shown in the picture, the trajectory diverged by 22.2 meters). From this we can deduce that with a lack of correspondences, SLAM fails to work.

By introducing a learnable model that fills in missing depth measurements, we may be able to circumvent this issue, obtain more accurate maps and achieve improvements in localization accuracy. In particular, by improving sparse measurements into denser maps, we might make it easier for the SLAM algorithm to find correspondences by increasing the number of points to work with. As we can clearly see from the pictures C.6a and C.6b, SLAM should work better with denser maps. The depth map improvement could be achieved by making the entire pipeline, including the SLAM module, differentiable and using a model suitable for depth completion.

Most depth completion architectures use Mean Square Error (MSE) or other, similar losses to penalize accuracy of depth completion. However, in our case, it might be beneficial to use map reconstruction error as a loss, since we want to primarily achieve improvement in this direction.

We have decided to use the KIITI dataset [5] to test our hypothesis as this dataset contains accurate ground truth dense depth data and also contains ground truth trajectory data, which are both needed for our experiments.

3.2 Description of the learning pipeline

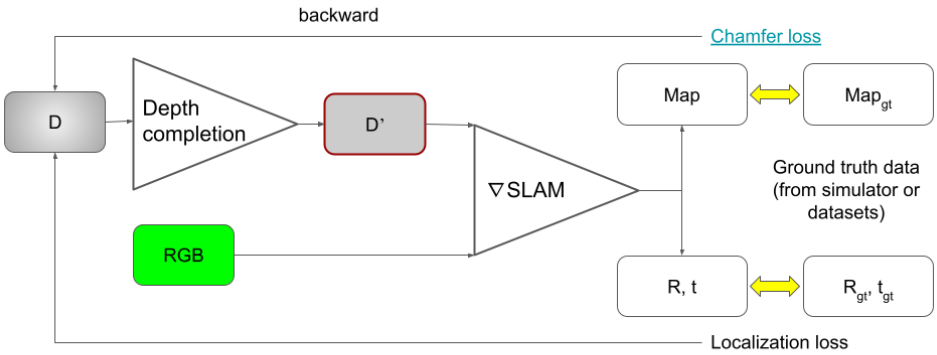


Figure (3.2) : Scheme of the depth completion pipeline with chamfer loss (provided by thesis supervisor)

In the picture 3.2 we can see a simplified scheme of the learning pipeline that we used. When training the model, we only used the chamfer loss, the

main reason being that the SLAM did not provide enough accurate pose data. Another reason is that we were limited by equipment because the gradSLAM module did not allow usage of other than the ground truth poses on the computational server.

The pipeline takes as input a sparse depth image (D) and passes this image through the depth completion model to create a predicted depth image D'. Predicted depth image together with the RGB image serves as input to the gradSLAM module. GradSLAM module then creates a map with ground truth trajectory. This map is then compared to the ground truth map, which is created also by the gradSLAM module, but using dense ground truth depth instead of D'. For map comparison, we used the chamfer distance from Pytorch3D (see 4.2.1) and then backpropagated the loss to update the model weights.

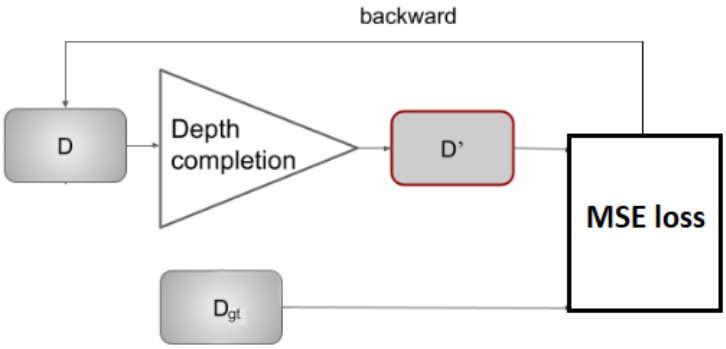


Figure (3.3) : Scheme of the depth completion pipeline with MSE loss (provided by thesis supervisor, edited)

A similar, simplified version of the pipeline was used when training with the MSE loss. The simplified scheme can be seen in the picture 3.3. As before, the input to the pipeline is the sparse depth image D, and this image is passed through the depth completion model. The predicted depth image D' is then compared with the dense depth image using the MSE loss (see 4.2.1). This loss is then again backpropagated to update the model weights.

3.3 Machine Learning Model

This work uses the Sparsity invariant DCNN machine learning model taken from the paper Sparsity Invariant CNNs [4]. This model was chosen because it performs well in the depth completion task and also had publicly available Pytorch implementation that was easy to add to an existing pipeline. Furthermore, the authors of this model tested it on the KITTI dataset, which we also decided to use for our work. Pytorch code of this model was taken from

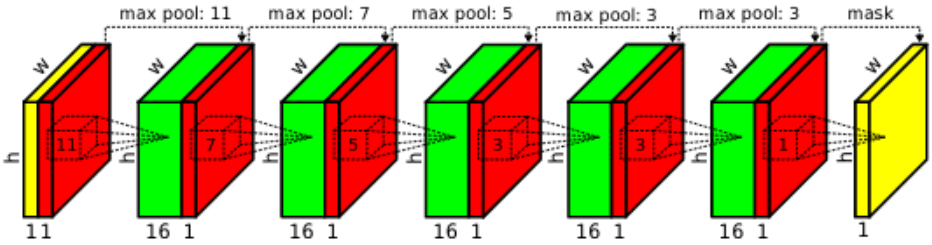


Figure (3.4) : Schematic of the network architecture [4]

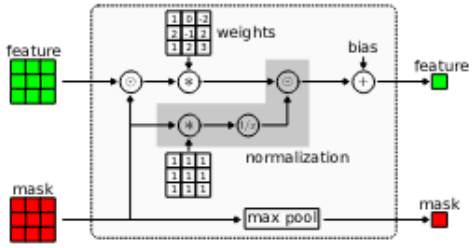


Figure (3.5) : Schematic of the sparse convolution [4]

[25]. The machine learning model was further modified to accommodate the same tensor shape as that used by the gradSLAM module.

The schematic of the model architecture can be seen in the picture 3.4. The input to the network is a sparse depth image (yellow) and a binary observation mask (red). The input is passed through several sparse convolution layers (dashed cubes) with decreasing kernel sizes from 11×11 to 3×3. The mask is passed through max pooling (max pool) for each sparse convolution layer.

The schematic of the sparse convolution can be seen in the picture 3.5. $\odot$ denotes elementwise multiplication, * convolution, 1/x inversion, and 'max pool' the max pool operation. The input feature can be single-channel or multi-channel. The sparse convolution operation explicitly considers sparsity by evaluating only observed pixels and normalizes the output appropriately. This helps to deal with unobserved inputs.

Chapter 4

Tools and Experiment Setup

4.1 Datasets and Data

4.1.1 ICL-NUIM

For the purpose of testing a demo version of the pipeline C.2 and testing the functionality and gradient propagation, the ICL-NUIM dataset was used [17]. This dataset has a few advantages because it is generated artificially. There are perfectly accurate trajectory poses, and the dataset was created as a benchmark for SLAM accuracy; therefore, it contains data suitable for SLAM and has an easy-to-work with structure. However, this dataset is probably not suitable for depth completion tasks, as there is no sparse data. We also need to address the issue of domain transfer when moving from simulated data to real data. Fortunately, the KITTI dataset has properties that made this transfer much easier.

4.1.2 KITTI

The main usability of SLAM tasks is in real-world environments such as caves, buildings, or urban areas. Therefore, we need to use real-world data for this research to be of any relevance. For this work, the KITTI dataset [5] was chosen. Heavily engineered to work for specific robots and environment systems KITTI allowed for easier transition to real data. Another advantages of KITTI are that it is a commonly used computer vision benchmark, contains a large amount of data, contains both depth and RGB images, and poses, which are all needed as input to SLAM.

We used two parts of the KITTI benchmark: the KITTI depth and raw data. KITTI depth contains more than 93 thousand samples that consist of dense and raw depth images and are aligned with the raw data [26]. Parts of the raw data that we use include RGB images, GPS data, and calibration data (specific for each drive). The KITTI raw dataset is divided into 5 dates (corresponding to the date the measurements were obtained), and each date

is further divided into separate drives (e.g. drive_0001). KITTI depth is divided into train and validation parts, where the validation part contains manually selected and cropped images. We decided not to use the validation data and only worked with the train data. The reason being that we would either require separate version of the model for the cropped validation data or we would need to crop training data in the same way the validation data is cropped. The structure of the validation dataset is also different. We manually divided the train drives into training, validation, and testing groups. In the picture 4.1 we can see how the constructed maps look visualized with Open3D. An example of a trajectory can be seen in the picture 4.2.

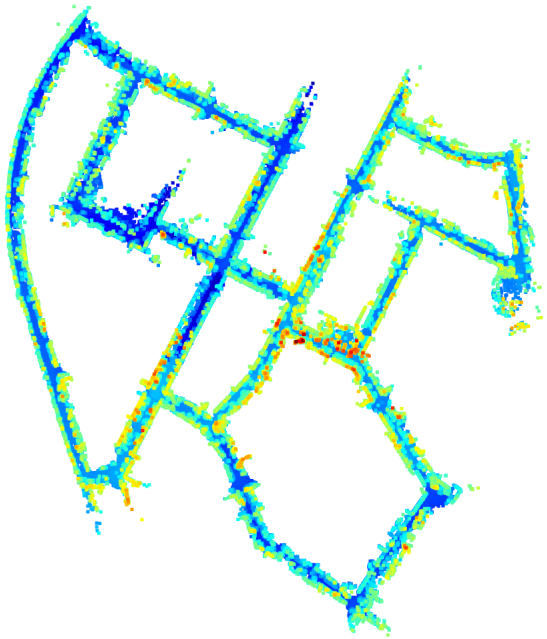


Figure (4.1) : Map created from dense depth frames of drive 10_03_0027_sync

The data was collected by driving a car (VW Passat station wagon) through different urban areas. The car was equipped with four video cameras (two color and two grayscale cameras), a rotating 3D laser scanner, and a combined GPS/IMU inertial navigation system.

The recording platform used two sets of cameras - on set on the left side of the car, the other on the right side, each side having a RGB and grayscale camera. For simplicity, we only used the left camera in our experiments.

The picture 4.4 illustrates the dimensions and mounting positions of the sensors (red) with respect to the body of the vehicle. Heights above the ground are marked in green and measured with respect to the surface of the road. Transformations between sensors are shown in blue. This figure also helps us to create the correct transformation from GPS coordinates into transformation matrices.

We used GPS data to create camera poses and then used these poses as ground truth. Note that the GPS data is not 100 % accurate. For

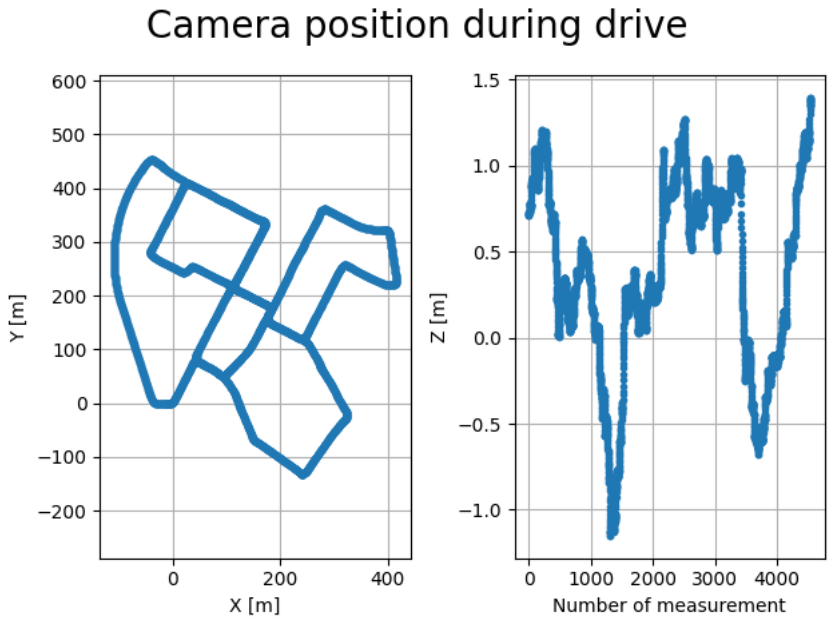


Figure (4.2) : Position of the camera during the drive 10_03_0027_sync

example, the elevation measurement would sometimes jump by 0.5. We use the 'synced' data that contains post-processed, rectified and synchronized video streams. In order to obtain the camera pose matrices, we have to apply several transformations on the poses of GPS in the world frame. These transformations can be seen in the picture 4.4.

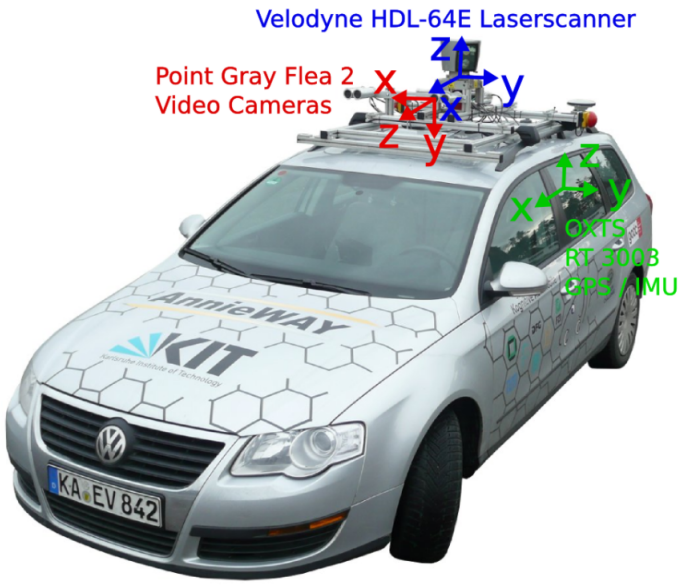


Figure (4.3) : Recording platform [5]

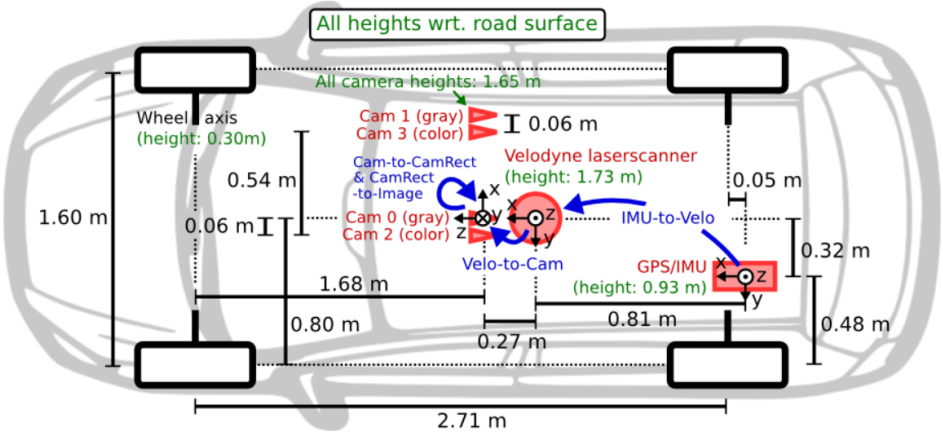


Figure (4.4) : Sensor setup [5]

4.2 Metrics

4.2.1 Loss Functions

Two loss functions were used: the chamfer distance and the mean square error (MSE). Chamfer distance is a common metric for measuring the similarity between two 3D point clouds. It is often used for its simplicity and comprehensibility. Chamfer distance between two sets of points $S_1, S_2 \subseteq \mathbb{R}^3$ can be defined as [9]:

$$d_{CD}(S_1, S_2) = |S_1|^{-1} \sum_{x \in S_1} \min_{y \in S_2} \|x - y\|_2^2 + |S_2|^{-1} \sum_{y \in S_2} \min_{x \in S_1} \|x - y\|_2^2$$

MSE measures the average square of per pixel differences. Let n be the number of values in the ground truth depth image, y_i be the pixel values of the predicted image and $\hat{y}_i$ be the pixel values of ground truth image, and i be the coordinate of the pixel in the ground truth image, then MSE can be compute as:

$$MSE = \frac{1}{n} \sum_{i=1}^{n} (y_i - \hat{y}_i)^2$$

4.2.2 Validation Criteria

Mean Absolute Error (MAE) and Root Mean Square Error (RMSE) are two of the most common metrics used to measure performance of depth completion. We decided to use these two metrics in this work as they have allowed us to better compare our results with other work and they provide understandable information about our depth completion performance.

We can easily compute these metrics from ground truth and predicted images. Let n be the number of values in the ground truth depth image, y_i be

the pixel values of the predicted image and $\hat{y}_i$ be the pixel values of ground truth image, and i be the coordinate of the pixel in the ground truth image, then the metrics can be computed as:

$$MAE = \frac{1}{n}\sum_{i=1}^{n}|y_i - \hat{y}_i|$$

$$RMSE = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2}$$

Both MAE and RMSE express average model prediction error in units of the variable of interest, in our case the metrics are expressed in millimeters, meaning MAE (or RMSE) of 1000 corresponds to an average difference between pixels of 1 meter. Both metrics can only take on positive numbers and are indifferent to the direction of errors. They are negatively-oriented scores, which means lower values are better, zero would correspond to no difference between ground truth and prediction.

Taking the square root of the average squared errors has some interesting implications for RMSE. Since the errors are squared before they are averaged, the RMSE gives a relatively high weight to large errors. Therefore, RMSE has the benefit of penalizing large errors more, and can be more appropriate in some cases [27]. In our work, penalizing outliers is important as points that are too far away can negatively influence point alignment during map reconstruction. However we decided to use both metrics to have more comprehensive evaluation.

4.2.3 Localization Accuracy

The poses of the robot are represented by transformation matrices which are comprised of a rotation matrix $\boldsymbol{R} \in SO(3)$ and a translation vector $\boldsymbol{t} \in \mathbb{R}^3$. Transformation matrix has the following form:

$$\boldsymbol{T} = \begin{bmatrix} & \boldsymbol{R} & & \boldsymbol{t} \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Localization accuracy is computed from two transformation matrices (be that $\boldsymbol{T_1}$ and $\boldsymbol{T_2}$) transformation matrices and comprises of two parts - translation distance and rotation angle. First we solve the delta transform between the two matrices, the matrix $\boldsymbol{X}$ corresponds to the transformation that we need to get from $\boldsymbol{T_1}$ to $\boldsymbol{T_2}$:

$$\boldsymbol{T_1 X} = \boldsymbol{T_2}$$

Then from the delta transformation matrix $\boldsymbol{X}$ we compute translation distance and angle of rotation. Translation distance is computed as the Frobenius norm of difference of the two translation parts:

$$d = \|\boldsymbol{t_X}\|_2$$

Rotation angle is computed as:

$$\alpha_x = acos((trace(\boldsymbol{R_X}) - 1)/2)$$

Finally the localization accuracy between two poses (L) is the sum of translation distance and rotaion angle:

$$L = d + \alpha_x$$

For two trajectories, both containing same amount of poses, we compute localization accuracy for each two corresponding poses, sum these values and divide by the number of poses in the trajectory. Localization accuracy corresponds to mean distance and rotation angle between each corresponding poses in the two trajectories.

4.3 Experiment setup

4.3.1 Model training

Several methods of training were proposed and implemented. The model was trained with the chamfer distance as a loss (chamfer loss) and separately with MSE as loss. We tried two approaches when training using the chamfer loss. First we trained with a map constructed from multiple frame (due to hardware requirement we were able to train with maps from a maximum of 8 frames) and then we trained with a map constructed only from a single frame. The reason we decided to use also just a single frame was to reduce the influence of dynamic objects. Another advantage was that this method required less memory to compute reconstruction loss and other point cloud operations. Two separate maps were constructed from corresponding images, one from dense ground truth images and the other from predicted depth images.

In order to introduce more diversity we used a different map in each iteration, for example in first iteration the map would be constructed from images 0-7 and in the second iteration it would be constructed from images 8-15. In order not to spend too much training time on the larger drives, the maximum number of used frames per drive was limited to 200. Once training on a drive finishes, we move to next one.

When training using the MSE loss, we tried two different approaches as well, these differed in how the MSE was computed. First we computed MSE only for pixels present in the sparse depth picture (we call this 'sparse masking') and second we tried computing MSE for pixels present in the dense depth image (we call this 'dense masking'). By pixels present in an image we mean pixels with value that in not NaN.

For all mentioned experiments we used the AdamW optimizer [28] with the following parameters: Learning rate = 0.001, Weight decay = 0.01. Using

this optimizer yielded better training results than SGD and Adam optimizers (loss was decreasing more rapidly, we did not conduct many tests with the other optimizers).

For both testing and training we used the Point fusion version of SLAM from the gradSLAM package, the main reason for this was that this was less memory intensive than using ICP-SLAM and thus we were able to use more frames for map construction. However the localization accuracy with ICP-SLAM was comparable to that of Point fusion.

Training was done 15 different drives with diverse landscapes and sizes.

Validation was done on different drives than the training process. For validation we used the exact same loop as for training, however we did not apply any backpropagation, only the forward pass. Four separate drives were used for validation and the validation metrics were averaged.

4.3.2 Model testing

In the testing process we create maps using the depth images and poses provided by the gradSLAM. We then compare the outputed trajectory to the ground truth trajectory computed from KITTI GPS data and IMU data. This is done for each of the depth image versions (sparse, dense, predicted). For computing the MAE and RMSE metrics we use the whole drives, and compare the predicted/sparse images to dense images.

The predicted depth images we first run through the model and saved, then the loop was run without the model with the loaded predicted images. We also did not compute chamfer distance and so we did not have to hold on to constructed maps. These factors enabled the testing loop to be run on significantly more frames than the training loop, localization accuracy results presented in the next chapter we computed from the first 30 frames of each drive.

Testing was carried out on five drives, different than training and validation. These drives are from the 'road' and 'city' environments and contain diverse surroundings.

4.4 Tools and Third Party Packages

4.4.1 Pytorch

The Pytorch library represents the backbone of the machine learning environment created for this work. It provides the data types used for the variables that are passed through machine learning networks as well as implementation of basic machine learning modules such as linear and convolutional layers. Pytorch also takes care of forward and backward passes and tweaking the

model weights when optimizing. We have chosen to work with Pytorch because it is compatible with the gradSLAM package.

Pytorch was written to allow efficient computation and provide a Python-friendly programming style consistent with other scientific libraries [29]. An important feature is also the support of usage of GPUs through the CUDA platform, which considerably speeds up computation.

4.4.2 CUDA

Compute Unified Device Architecture or CUDA is a parallel computing platform developed by Nvidia that allows software to use certain types of graphics processing units (GPUs) for general-purpose processing [30]. The CUDA toolkit distributed by Nvidia [31] includes GPU-accelerated libraries, debugging and optimization tools, a C/C++ compiler, and a runtime library that allow users to build and deploy applications on major architectures including x86, Arm and POWER.

Using computational power of GPUs allows neural network related computations (like backpropagation) to be performed at much higher speed. This makes CUDA an indispensable tool for any machine learning process. Training machine learning models on GPUs can be several orders of magnitude faster than training on CPU and for larger models it can be the only feasible way of training them.

4.4.3 GradSLAM

In our pipeline, gradSLAM serves as the module that performs SLAM, making it an essential part of the pipeline and allowing the entire pipeline to be differentiable and to backpropagate the gradient through SLAM.

Gradslam is described by its authors [11] as a fully differentiable dense SLAM system, with the central idea being the construction of a computational graph that represents each operation in a dense SLAM system. The authors make the module differentiable by proposing separate differentiable alternatives to several non-differentiable components of traditional dense SLAM systems, such as optimization, odometry estimation, raycasting, and map fusion. This creates a pathway for the gradient flow from 3D map elements to sensor observations (e.g., pixels). This package implements differentiable variants of two dense SLAM systems that operate on voxels, surfels, and pointclouds, respectively. These variants are ICP-SLAM and Point Fusion.

Central to our goal of realizing a fully differentiable SLAM system are computational graphs, which underlie most gradient-based learning techniques. We make the observations that, if an entire SLAM system can be decomposed into elementary operations, all of which are differentiable, we could compose these elementary operations to preserve differentiability. However, modern dense SLAM systems are quite sophisticated, with several non-differentiable

subsystems (optimizers, raycasting, surface mapping), that make such a construct challenging.

4.4.4 Pytorch3D

Pytorch3D is a library that provides differentiable solutions for 3D data related operations [9]. It is built, as the name might suggest, on the Pytorch library and it is fully compatible with it. On top of that, all operations implemented in Pytorch3D come with CUDA support.

Pytorch3D package was used to calculate the chamfer distance and differentiable transformations between sensor poses on the GPU. For a detailed explanation of what the chamfer distance is and how it is computed, see the section describing loss for the training model 4.2.1. Localization accuracy was computed with differentiable pose transformations and therefore could be also used as a loss function.

4.4.5 ROS and Rviz

The Robot Operating System (ROS) is an open source software that defines the tool, interfaces, and components for building robotic units [32]. ROS allows programmers to connect actuators, sensors, and control units through tools called topics and messages. Additionally, ROS can run on real robots and also in simulators. We did not conduct any tests on real robots, however ROS allowed us to run SLAM in simulator and better visually understand how it performed. ROS also supports Python, which made it convenient for us to use.

The Rviz simulator, which is based on ROS, was used for real-time simulation of the map construction process, allowing us to see how the map was constructed and where the SLAM struggled or what the differences were between sparse, dense, and predicted depth data. We also visualized the ground truth and computed trajectories and computed localization accuracy from them.

Chapter 5

Results

To get a sense of what kind of experiments have been performed, please refer to section 4.3. The explanation of the metrics presented in this chapter can be found in subsections 4.2.1, 4.2.2 and 4.2.3.

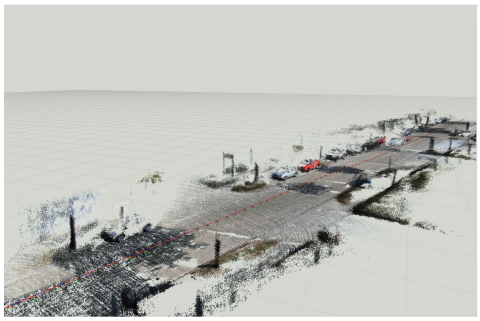
5.1 Training results

Table 5.1 shows the best metrics achieved during the validation process. The MSE value for loss with a sparse mask can seem a bit off compared to other values. This happens because the values from which it was computed are not the same as those of the other metrics that used a dense mask.

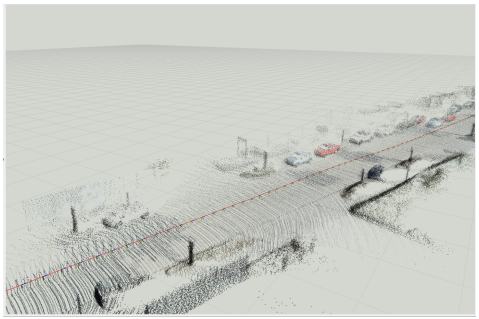
Training method	Chamfer distance [m]	MSE [m]	MAE [mm]	RMSE [mm]
Performance of models trained with chamfer loss				
Using map from single frame	**1.93**	✗	471	**1206**
Using map from multiple frames	4.78	✗	753	1476
Performance of models trained with MSE loss				
Using a sparse mask loss	✗	34.73	9030	10638
Using a dense mask loss	✗	**2.23**	**400**	1316

Table (5.1) : Comparison of metrics for validation of a model trained with chamfer loss and MSE loss (lower is better)

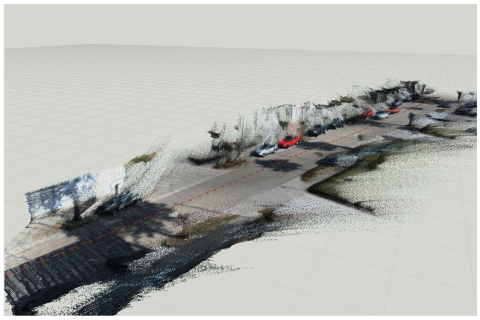
Graphs detailing the loss during training and validation can be seen in the figure 5.1. From these images we can see that the map created from the predicted data is denser than even the dense ground truth data. We can better recognize objects like cars and the surfaces like road are much smoother. However there are noticeable artifacts are lines of trailing points. Constructed maps with different types of losses and depth completion examples can be seen in the Appendix C.



(a) : Map constructed from dense ground truth data



(b) : Map constructed from sparse depth data



(c) : Map constructed from data predicted data

Figure (5.1) : Comparison of maps constructed with ground truth poses

5.2 Testing results

The tests presented in table 5.2 were performed on the first 30 subsequent samples of each drive. The values in the table are averages of the values computed on these drives. Note that MAE and RMSE were computed only for pixels present in the dense depth image, and the MSE sparse depth mask loss uses only pixels present in the sparse depth image, thus these values do seem off in the table. The MAE and RMSE metrics were not presented for the dense depth images, since they served as the ground truth to compare with.

We also add a table with results published by the authors of the model 5.3 for comparison with our results. Note that these results were obtained on the KITTI depth selection, which contains the handpicked images for validation and testing, so our validation and testing datasets are different. MAE and RMSE presented in this table are computed in the same way as we did. We achieved comparable results to the original work and moreover, we made a step forward, by using the model for evaluation of SLAM.

Training method	Localization accuracy [-]	MAE [mm]	RMSE [mm]
Performance of models trained with chamfer loss			
Using map from single frame	10.36	561	1774
Using map from multiple frames	10.34	621	1810
Performance of models trained with MSE loss			
Using a sparse mask loss	10.31	10639	13182
Using a dense mask loss	10.29	**536**	**1771**
Performance without a model			
Using sparse depth	9.447	14662	19524
Using dense depth	**3.76**	✗	✗

Table (5.2) : Comparison of metrics averaged across testing drives (lower is better)

Mode	MAE [mm]	RMSE [mm]
Validation	680	2010
Testing	540	1810

Table (5.3) : Comparison of metrics published by the model authors [4]

Chapter 6

Discussion

The hypothesis presented in the section 3.1 was not confirmed as the localization accuracy did not improve by introducing a depth completion model, and it even got worse than when using sparse depth data. From the results presented in the appendix C, we can see that the depth completion works, and the completed images look like a denser version of the ground truth depth images. However, when we construct a map from the predicted depth images, it becomes clear that there is a large number of outliers present and we can see ray-like structures. The model tries to finish the whole picture, and sometimes it fills in the gaps between points with what could be considered to be a far background. These artifacts are probably a feature of the model, since they were introduced with every type of loss that we used.

It was realized that it is possible to use the chamfer distance as a novel way of training depth completion networks. This is supported by the testing results in table 5.2, where we can see that the model trained with chamfer loss only got slightly worse metrics than the model trained with MSE loss, and in the validation results (5.1), we were even able to get the RMSE lower with chamfer loss.

From the results, it is clear that training using a chamfer loss with a map constructed with only one frame allows us to obtain better metrics and lower loss. However, we did not see any substantial improvement in localization accuracy. The better performance of the single frame map might be because the KITTI dataset contains a lot of dynamic objects (moving cars, cyclists, etc.) and these cause the introduction of additional artifacts and noise for the model.

The metrics we obtained with the used model are very close to the metrics that the authors of the used model present in their paper [4]. This probably means that the model was trained correctly and that there was no significant error in the training loop, at least in the backpropagation.

The test values have differed for each drive. Reconstructing maps proved especially difficult on drives with fewer features and open terrain (e.g. drive 09 26 0001) as these made it harder to acquire correct point correspondences. Unfortunately, the GradSLAM package did not allow for other than ground

truth odometry to be used on the computational server, and it was not feasible to run testing loop locally on more drives for all differently trained models trained with different losses.

Chapter 7

Conclusion

A depth completion network [4] was implemented and trained using map reconstruction accuracy (chamfer distance). We used a pipeline that included a differentiable dense SLAM module [11]. We trained four networks with two losses, each loss with two variants. We were able to obtain denser depth data and create more detailed maps.

The results showed that training with map reconstruction error is a viable way of training depth completion networks, as the metrics were similar to the original solution. We were able to improve the mapping part of SLAM and create better maps. However, introducing the model did not yield any improvement in localization accuracy. This was due to the fact that the model created a large number of outliers.

7.1 Future work

The machine learning model used in this work introduced many outliers, and this, in turn, probably negatively impacted the performance of SLAM. A large number of artifacts or outlier points might make it difficult for SLAM to find the correct correspondences, and thus the reconstructed map is not accurate. These outlier points were present in all trained versions of the model, and thus this might be a feature of the model architecture. A good direction to focus future work would be to try a pipeline with a different model, perhaps a model that uses as input not only depth but RGB images as well.

An interesting experiment to try would be also to introduce some kind of pointcloud filtration method. We briefly tried hidden point filtration, but due to not enough time remaining, we did not conduct any detailed experiments. Another method to explore would be to use localization error to fine-tune a model pre-trained with reconstruction error.

Appendix A

Bibliography

[1] Karel Zimmermann. Aro, lecture 3 - geometry slam from lidar and camera, Mar 2021.

[2] Maik Keller, Damien Lefloch, Martin Lambers, Shahram Izadi, Tim Weyrich, and Andreas Kolb. Real-time 3d reconstruction in dynamic scenes using point-based fusion. In *2013 International Conference on 3D Vision - 3DV 2013*, pages 1–8, 2013.

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

[4] Jonas Uhrig, Nick Schneider, Lukas Schneider, Uwe Franke, Thomas Brox, and Andreas Geiger. Sparsity invariant cnns. In *2017 International Conference on 3D Vision (3DV)*, pages 11–20, 2017.

[5] Andreas Geiger, Philip Lenz, Christoph Stiller, and Raquel Urtasun. Vision meets robotics: The kitti dataset. *International Journal of Robotics Research (IJRR)*, 2013.

[6] Wenyuan Zeng, Wenjie Luo, Simon Suo, Abbas Sadat, Bin Yang, Sergio Casas, and Raquel Urtasun. End-to-end interpretable neural motion planner, 2021.

[7] Vojtech Salansky, Karel Zimmermann, Tomas Petricek, and Tomas Svoboda. Pose consistency kkt-loss for weakly supervised learning of robot-terrain interaction model. *IEEE Robotics and Automation Letters*, 6:5477–5484, 07 2021.

[8] Brandon Amos, Ivan Dario Jimenez Rodriguez, Jacob Sacks, Byron Boots, and J. Zico Kolter. Differentiable mpc for end-to-end planning and control, 2018.

[9] Nikhila Ravi, Jeremy Reizenstein, David Novotny, Taylor Gordon, Wan-Yen Lo, Justin Johnson, and Georgia Gkioxari. Accelerating 3d deep learning with pytorch3d, 2020.

[10] Peter Karkus, Shaojun Cai, and David Hsu. Differentiable slam-net: Learning particle slam for visual navigation, 2021.

[11] Krishna Murthy Jatavallabhula, Ganesh Iyer, and Liam Paull. ∇slam: Dense slam meets automatic differentiation. In *2020 IEEE International Conference on Robotics and Automation (ICRA)*, pages 2130–2137, 2020.

[12] Richard A. Newcombe, Shahram Izadi, Otmar Hilliges, David Molyneaux, David Kim, Andrew J. Davison, Pushmeet Kohi, Jamie Shotton, Steve Hodges, and Andrew Fitzgibbon. Kinectfusion: Real-time dense surface mapping and tracking. In *2011 10th IEEE International Symposium on Mixed and Augmented Reality*, pages 127–136, 2011.

[13] Thomas Whelan, Michael Kaess, Hordur Johannsson, Maurice Fallon, John J Leonard, and John McDonald. Real-time large-scale dense rgb-d slam with volumetric fusion. *The International Journal of Robotics Research*, 34(4-5):598–626, 2015.

[14] Jakub Paplhám. Convolutional neural networks with local context masks. 2020.

[15] Haofei Xu and Juyong Zhang. Aanet: Adaptive aggregation network for efficient stereo matching. *CoRR*, abs/2004.09548, 2020.

[16] Xuelian Cheng, Yiran Zhong, Mehrtash Harandi, Yuchao Dai, Xiaojun Chang, Tom Drummond, Hongdong Li, and Zongyuan Ge. Hierarchical neural architecture search for deep stereo matching, 2020.

[17] Ankur Handa, Thomas Whelan, John McDonald, and Andrew J. Davison. A benchmark for rgb-d visual odometry, 3d reconstruction and slam. In *2014 IEEE International Conference on Robotics and Automation (ICRA)*, pages 1524–1531, 2014.

[18] Joseph White. Waymo opens driverless robo-taxi service to the public in phoenix, Oct 2020.

[19] Darpa subterranean challenge. `https://www.subtchallenge.com/`.

[20] Sebastian Thrun, Wolfram Burgard, and Dieter Fox. *Probabilistic Robotics (Intelligent Robotics and Autonomous Agents)*. The MIT Press, 2006.

[21] Jana Procházková and Dalibor Martišek. Notes on iterative closest point algorithm. 04 2018.

[22] P.J. Besl and Neil D. McKay. A method for registration of 3-d shapes. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 14(2):239–256, 1992.

[23] Tommi Tykkälä and Andrew I. Comport. A dense structure model for image based stereo slam. In *2011 IEEE International Conference on Robotics and Automation*, pages 1758–1763, 2011.

[24] Mu Hu, Shuling Wang, Bin Li, Shiyu Ning, Li Fan, and Xiaojin Gong. Penet: Towards precise and efficient image guided depth completion. *CoRR*, abs/2103.00783, 2021.

[25] chenxiaoyu523. Sparsity-invariant-cnns-pytorch. `https://github.com/chenxiaoyu523/Sparsity-Invariant-CNNs-pytorch`, 2019.

[26] The kitti vision benchmark suite. `http://www.cvlibs.net/datasets/kitti/index.php`.

[27] Mae and rmse — which metric is better? `https://medium.com/human-in-a-machine-world/mae-and-rmse-which-metric-is-better-e60ac3bde13d`, March 2016.

[28] Ilya Loshchilov and Frank Hutter. Decoupled weight decay regularization, 2017.

[29] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Köpf, Edward Yang, Zach 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, 2019.

[30] Fedy Abi-Chahla. Nvidia's cuda: The end of the cpu? `https://www.tomshardware.com/reviews/nvidia-cuda-gpu,1954.html`, June 2008.

[31] Nvidia developer. `https://developer.nvidia.com`.

[32] Ros - robot operating system. `https://www.ros.org/`.

[33] Sachin Mohan. Bilateral filtering in python opencv with cv2.bilateralfilter(). `https://machinelearningknowledge.ai/bilateral-filtering-in-python-opencv-with-cv2-bilateralfilter/`, December 2020.

[34] Kai Zhang, Wangmeng Zuo, Yunjin Chen, Deyu Meng, and Lei Zhang. Beyond a gaussian denoiser: Residual learning of deep cnn for image denoising. *IEEE Transactions on Image Processing*, 26(7):3142–3155, 2017.

Appendix B

Training and validation loss

Graphs in this may lack the training value for the 0th episode, as it was much larger than the rest of the values and caused distortion in the graphs. The validation value is computed only after the 0th training episode, so the value is lower than that for the initialized model. Additionally, training loss can sometimes seem to jump higher or lower. This occurred at times when a new training sequence was introduced.

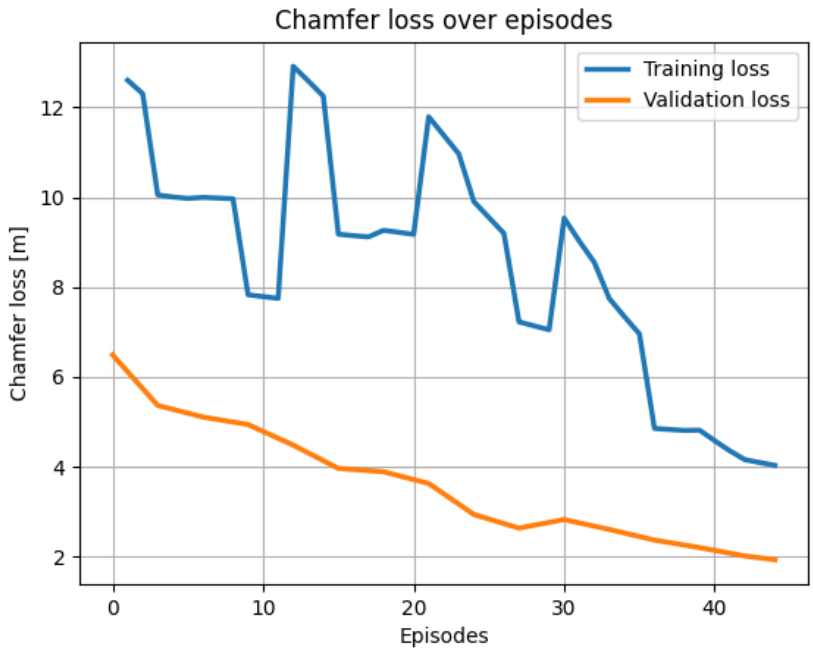


Figure (B.1) : Chamfer loss during training with a map constructed from a single frame

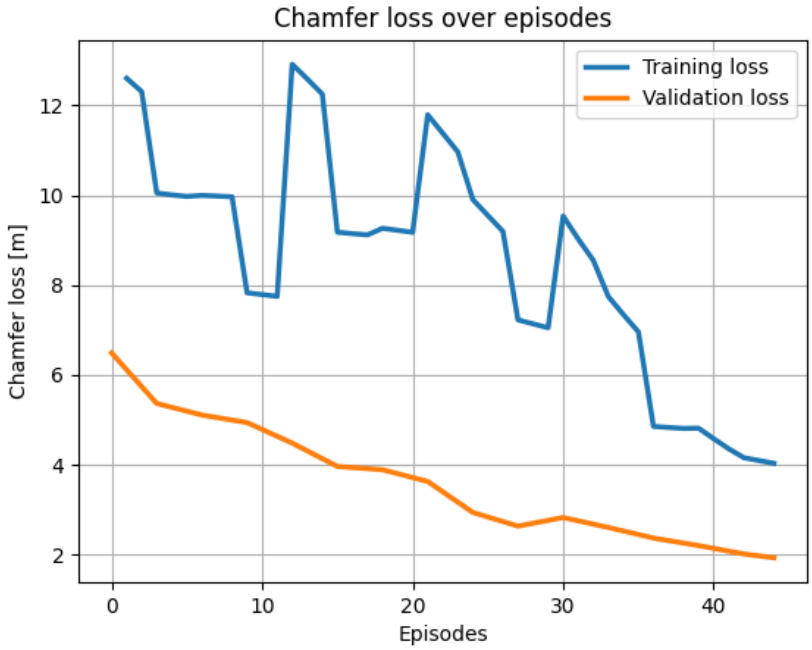


Figure (B.2) : Chamfer loss during training with a map constructed from a single frame

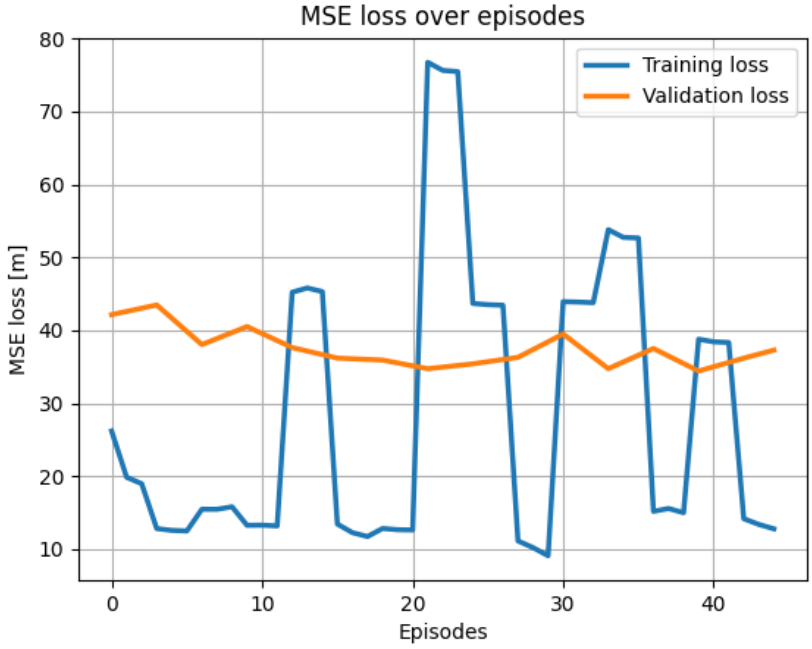


Figure (B.3) : MSE during training with MSE loss with a sparse mask

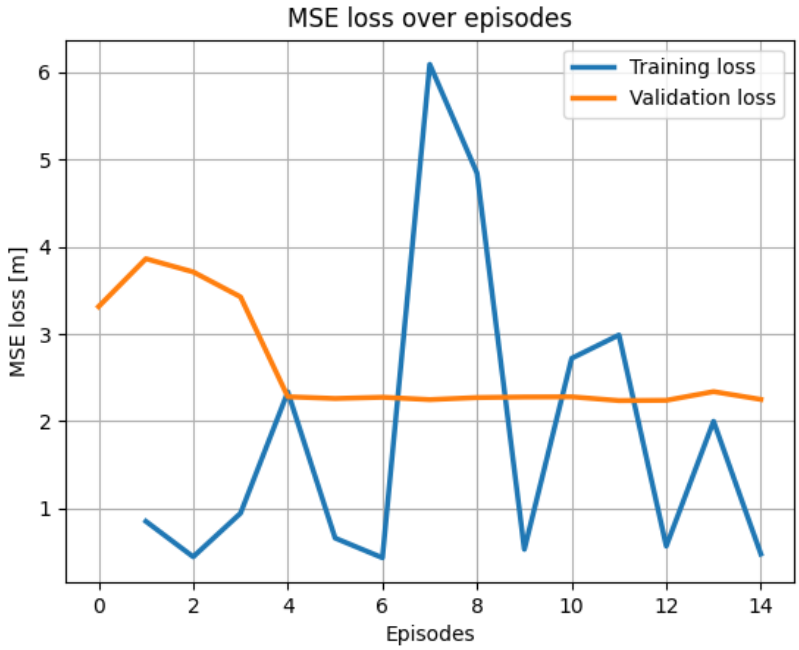


Figure (B.4) : MSE during training with MSE loss with a dense mask

Appendix C

Depth completion results

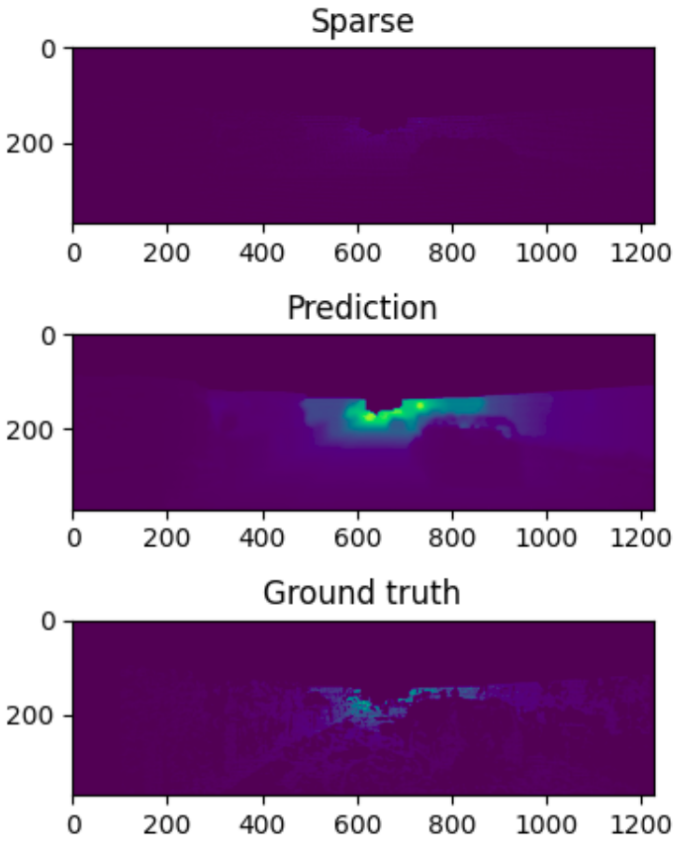


Figure (C.1) : Example of depth completion with a model trained with chamfer distance

.

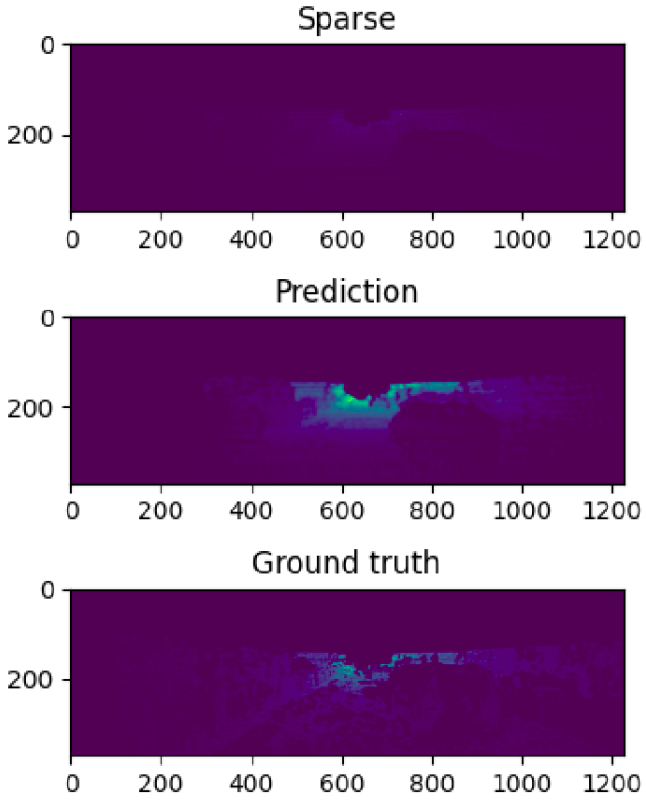


Figure (C.2) : Example of depth completion with a model trained with MSE loss with a sparse mask

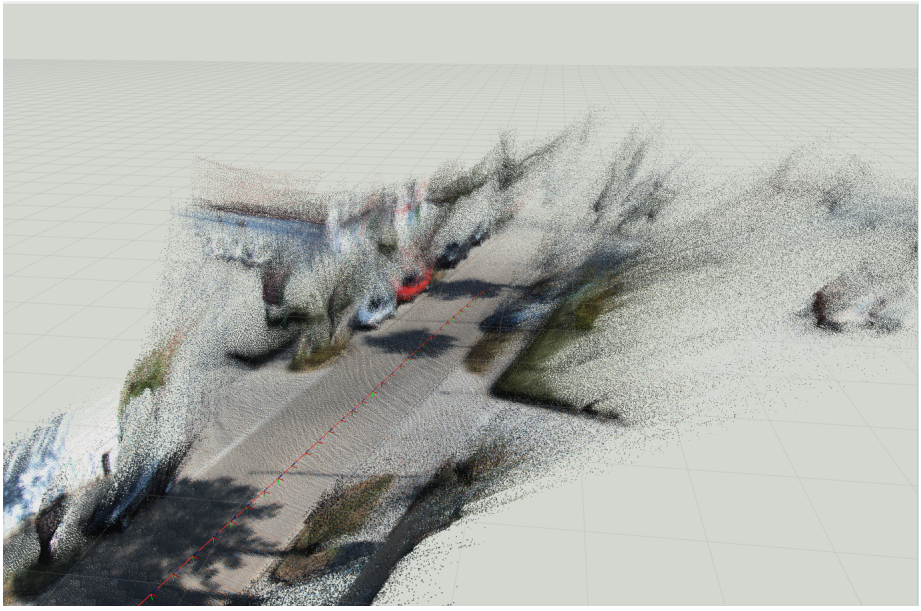


Figure (C.3) : Example of a map constructed from data predicted with a model learned with MSE loss with a dense mask

C.1 Depth filtration

We utilized two methods of depth filtration on the predicted depth data in hopes, that we would reduce the number of outlier points. First we tried removing point that lay too close or too far away from the camera (image C.5). Second we tried using a bilateral filter [33] to average out depth values with neighboring pixels (image C.4). . Both of these filtration methods did not bring any improvement in localization accuracy that would be worth mentioning. This leads to the conclusion that the depth completion model does not provide reliable enough measurements that could be used for the alignment process, even when using the mentioned filters.

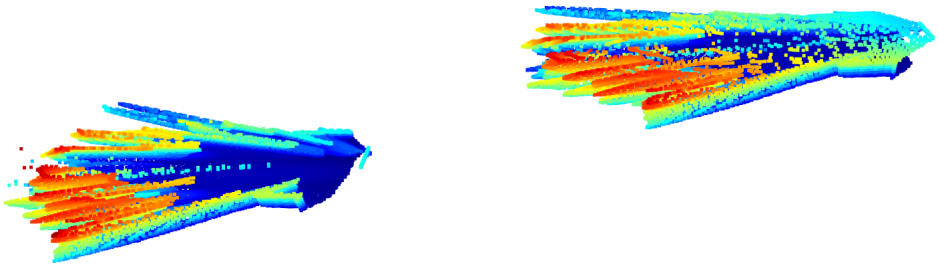
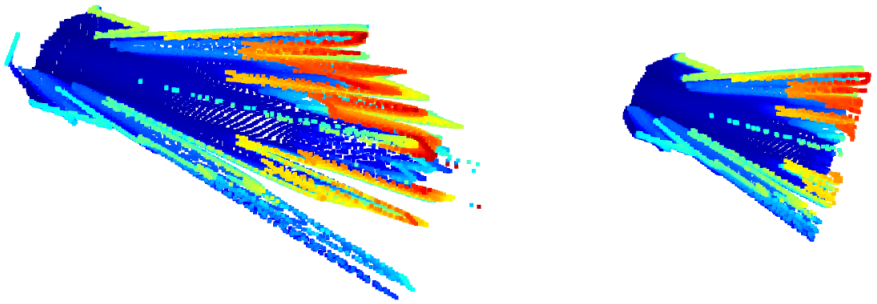


Figure (C.4) : Depth filtration using openCV bilateral filter (filtered pointcloud in on the right)



s

Figure (C.5) : Depth filtration by removing far laying points (filtered point-cloud in on the right)

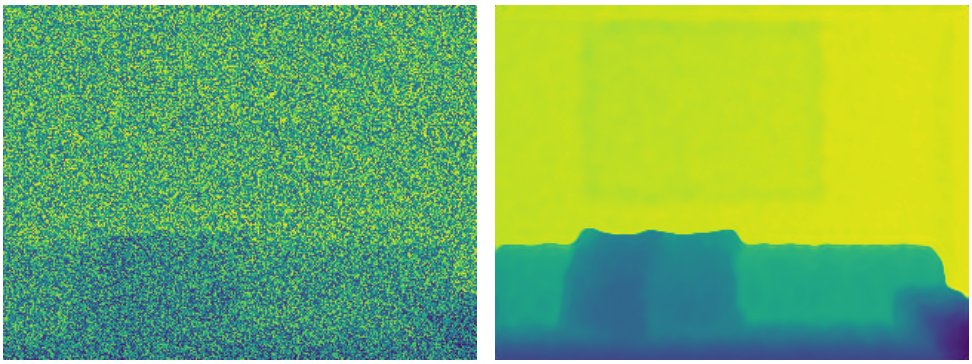
C.2 Demo version using perfect data

For the purposes of testing a demo version of the pipeline was used on the ICL-NUIM dataset (see 4.1.1). This perfect artificial data did not allow for

obtaining real world applicable results, but was still useful in showing whether gradients can get propagated. The general task of this demo version was to find out whether it was possible to achieve convincing results on removing noise from depth images using the chamfer distance between ground truth and reconstructed pointclouds as a loss.

We have used a denoising convolutional neural network taken from the paper Beyond a Gaussian Denoiser: Residual Learning of Deep CNN for Image Denoising [34]. Noisy images we constructed from ground truth images with artificially added Gaussian noise. In the pipeline, we first constructed a point cloud from the image that passed through the model and then a second pointcloud from the ground truth image. We computed the chamfer distance (loss) between these two pointclouds and backpropagated to update the model weights.

The results showed that it is indeed possible to use the chamfer distance computed from pointclouds constructed with the gradSLAM module for learning. The gradients flowed through the pipeline and the model learned to denoise depth images as we can see in figure C.6.



(a) : Depth image with added Gaussian noise

(b) : Depth image after passing through model

Figure (C.6) : Comparison of noisy and denoised images

BACHELOR‘S THESIS ASSIGNMENT

I. Personal and study details

Student's name: **Staněk Jáchym** Personal ID number: **492374**

Faculty / Institute: **Faculty of Electrical Engineering**

Department / Institute: **Department of Cybernetics**

Study program: **Cybernetics and Robotics**

II. Bachelor’s thesis details

Bachelor’s thesis title in English:

Machine Learning for Robotic Exploration

Bachelor’s thesis title in Czech:

Strojové učení pro robotickou exploraci

Guidelines:

Goal: correction of sensory measurements that are input to SLAM (Simultaneous Localization and Mapping) system.
Sensory data: RGB-D (depth) images.
SLAM system: GradSLAM, differentiable implementation of PointFusion (ICP-SLAM).
The core of the depth correction (completion) pipeline is the differentiable SLAM module, which takes as input RGB-D images and outputs camera trajectory and map estimate. However, the input sensory data could have noise and missing depth values.
Therefore it is important to introduce a Depth Correction (DC) module, before the propagation of the depth measurements through the SLAM module. The DC module could also take as input another data (normals). DC and gradSLAM are differentiable modules, which should allow propagating gradients through the entire pipeline from input sensory data to the resultant map and camera trajectory. Once we have a constructed map (we can use ground truth trajectory if it is available), we can calculate the loss:
(1) reconstruction error (for example Chamfer distance between constructed and truth point clouds) using the ground truth map (available in datasets or in Subt simulator),
(2) localization error (using the ground truth poses from datasets or the simulator).
Once the loss is computed, the weights of the DC module could be updated (iterative process). We run the optimization process (iterations) until satisfied with the reconstruction error.

Bibliography / sources:

[1] K. Murthy J., et al, gradSLAM: Automagically differentiable SLAM, ICRA, 2020
[2] P. Karkus, et al, Differentiable SLAM-net: Learning Particle SLAM for Visual Navigation, CVPR, 2021
[3] D. S. Chaplot, et al, Active Neural Localization, ICLR, 2018
[4] S. K. Gottipati, et al, Deep Active Localization, RAL, 2019
[5] Thomas M Howard and Alonzo Kelly, Optimal rough terrain trajectory generation for wheeled mobile robots, The International Journal of Robotics Research, 2007
[6] Vojtěch Šalanský, et al, Pose consistency kkt-loss for weakly supervised learning of robot-terrain interaction model, RAL, 2021
[7] Zichao Zhang and Davide Scaramuzza, Fisher information field: an efficient and differentiable map for perception-aware planning, arXivpreprint arXiv:2008.03324, 2020.

Name and workplace of bachelor's thesis supervisor:

MSc. Ruslan Agishev Vision for Robotics and Autonomous Systems FEE

Name and workplace of second bachelor's thesis supervisor or consultant:

Date of bachelor's thesis assignment: **03.12.2021** Deadline for bachelor thesis submission: **20.05.2022**

Assignment valid until: **30.09.2023**

MSc. Ruslan Agishev Supervisor's signature	prof. Ing. Tomáš Svoboda, Ph.D. Head of department's signature	prof. Mgr. Petr Páta, Ph.D. Dean's signature

III. Assignment receipt

The student acknowledges that the bachelor's thesis is an individual work. The student must produce his thesis without the assistance of others, with the exception of provided consultations. Within the bachelor's thesis, the author must state the names of consultants and include a list of references.

Date of assignment receipt	Student's signature