CHETAN SRINIVASA KUMAR, B.Eng

URBAN VISUAL LOCALIZATION WITH MAP DATA

Master's Thesis

to achieve the university degree of

Master of Science

Master's degree programme: Computer Science

submitted to

Graz University of Technology

Supervisor

Assoc.Prof. Dipl-Ing. Dr.techn. Friedrich Fraundorfer

Institute of Computer Graphics and Vision

Graz, January 2021

This document is set in Palatino, compiled with pdfLaTeX2e and Biber.

The LaTeX template from Karl Voit is based on KOMA script and can be found online: https://github.com/novoid/LaTeX-KOMA-template

Affidavit

I declare that I have authored this thesis independently, that I have not used other than the declared sources/resources, and that I have explicitly indicated all material which has been quoted either literally or by content from the sources used. The text document uploaded to TUGRAZonline is identical to the present master's thesis.

Date	Signature

Abstract

Visual localization is the problem of estimating the camera pose of a given image with respect to the representation of a known scene. We focus on its use in the positioning of a car driving through an urban area. Traditional methods of localization often require considerable resources to construct representations of the known scene (point clouds in Structure from Motion, for instance). To circumvent this problem, we propose a new method where we utilise the universally available OpenStreetMap map tiles of an urban area to train a pose regression network. To infer the pose at some scene in that area, we extract an OpenStreetMap tile-like representation using the scene images acquired from cameras, and feed it to the aforementioned pose regression network. We outline our method in detail, and then demonstrate its feasibility by evaluating on locations drawn from a driving sequence of the Oxford Robotcar dataset.

Acknowledgement

I'd like to extend my sincere thanks to my supervisor Assoc. Prof. Dipl.-Ing. Dr.techn. Friedrich Fraundorfer for giving me the opportunity to complete my thesis under his most capable guidance at the Institute of Computer Graphics and Vision, and to my advisor Dipl.-Ing Sinisa Stekovic for his support and helpful brainstorming sessions during the course of this thesis. I owe my thanks to my colleagues at the ICG who were always ready with feedback or just to bounce ideas around. Finally, my gratitude to my parents and brother cannot be overstated for their unceasing support for the entire duration of my studies at the University.

Contents

List of Figures

1 Introduction

1.1 Motivation

Visual localization is the idea of extracting the pose of a camera given an image captured from it, with respect to some representation of a known scene.

Visual localization is vital to applications that require robust navigation capabilities. An example that comes to mind would be the popular Autonomous Driving paradigm, wherein modalities such as GPS can be unreliable in dense urban areas. The camera, therefore, becomes a viable option to provide accurate estimates of the current position.

The standard way to tackle this problem would be to utilize Structure from Motion(SfM). SfM utilizes corresponding points between two images and the geometry between two views to construct a 3D pointcloud of the scene. A query image's location is obtained by finding points of the 3D pointcloud that match with image points, and utilizing geometric techniques to fit a pose to these correspondences. SfM, however, does not scale very well to city-scale areas where obtaining suitable image data and a usable reconstruction are a major problem. It also does not behave well with later modifications to the structure of its representation.

A quicker but less robust way is to store a database of images of known locations in the required area, indexed by carefully chosen image-specific features. The images closest to the query image would be extracted using these features, and an approximate location could be interpolated. This approach is definitely more scalable, but is susceptible to false positives and failure if the sampling of images in an area are not dense enough.

Deep Learning's rise has affected Visual Localization as well. The seminal paper on PoseNet [KGC15], proved that Localization is possible by training a Convolutional Neural Network (CNN) with image and pose data. This approach definitely mitigates both the scale and accuracy problems, but the work by [Sat+19], proves that PoseNet variants are not as powerful as traditional SfM approaches. He proves that the PoseNet variants end up learning several "basis" poses, of which the prediction is a linear combination.

State-of-the-art methods have made strides in mitigating this problem.

We propose a method that utilizes a CNN to infer pose directly on the map raster of the area in question, given a 2D layout of buildings visible from a traveling car. We aim to address convincingly the problem of data acquisition, and scale of localization with this method.

1.2 Contribution

In this thesis, we estimate a coarse localization estimate of a car traversing through an urban setting.

We reformulate the visual localization problem as a 2d pose estimation directly on the map raster. Our Localization CNN ([Bra+17], MapNet) is trained on OSM map tiles of an area, where the pose of a tile is simply a 2D offset from some arbitrary reference tile.

At test time, the trained network outputs the pose from the input map tile representation obtained from the pointcloud of the scene.

The method opens up a new data source for training visual localization networks - OpenStreetMap tile data, which covers most of the known urban world. We show that we can get reasonable pose estimates using just building/road pointclouds, using MapNet trained over large sections of a city.

We evaluate the method with the Oxford RobotCar sequence. We choose locations in the drive sequence which have distinct building shapes, and directly infer GPS locations from the MapNet trained on that section of Oxford.

1.3 Outline

We begin by describing the core concepts utilized in the method outlined in this thesis: the basic concepts of Structure from Motion(SfM), and a delineation of neural networks and convolutional neural networks (CNNs). We then describe the traditional solution to the localization problem, and proceed to the seminal localization CNN. We later describe MapNet, and how we utilize it for our method. The process of getting from the PointCloud of the scene to a 2D Map-Tile representation ready for query is also described in this chapter. We then describe how we generate the dataset with which we train MapNet from OpenStreetMap.

Finally, we showcase our experiments and results and close with our conclusions.

2 Theoretical Background

2.1 The Basics of Structure from Motion

In this section, we review the fundamental principles of the SfM pipeline. The SfM method is the classical solution to the visual localization problem, and the diagram 2.1 outlines the basic steps of the SfM pipeline.

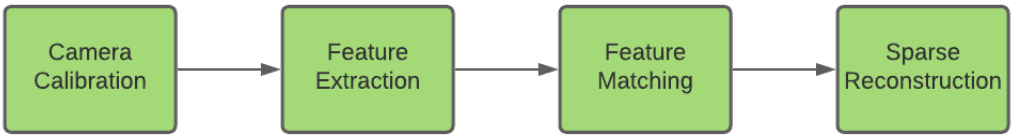


Figure 2.1: Simplified SfM Pipeline

We shall describe the basic ideas of each stage of the pipeline, as relevant to the localization problem.

2.1.1 Camera Calibration

Consider the case of projecting a point in 3D to a the image 2D plane, outlined in the figure 2.2. The 2D point $\mathbf{x}$ can be described by the ray $\lambda[u\ v\ 1]^T$ in projective space.

The 3D point $\mathbf{x}$ can be projected onto the image plane by a projection matrix $\mathbf{P}$, thus:

$$\mathbf{x} = \mathbf{PX} = \mathbf{K}[\mathbf{R}|\mathbf{t}]\mathbf{X}$$

Note the decomposition of the projection matrix $\mathbf{P}$ as $\mathbf{K}[\mathbf{R}|\mathbf{t})$, wherein $\mathbf{K}$ is called the intrinsic matrix. $[\mathbf{R}|\mathbf{t}]$ is a concatenation of the rotation and translation of the camera w.r.t some world coordinate system, called the extrinsic matrix.

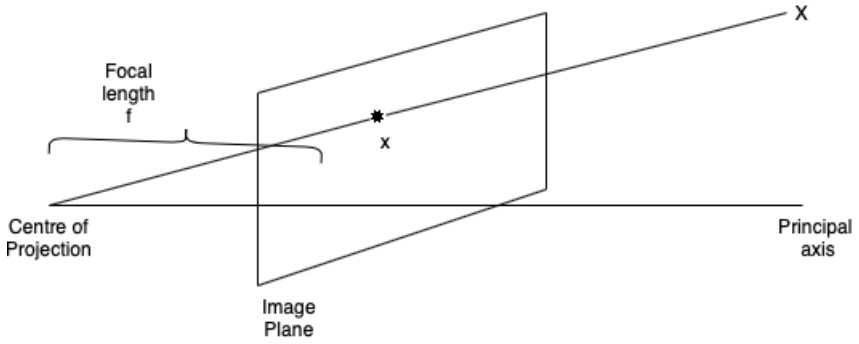


Figure 2.2: Pinhole Camera. X is the 3D point, x is the 2D point.

We model $\mathbf{K}$ as follows:

$$\mathbf{K} = \begin{bmatrix} f & s & c_x \\ 0 & af & c_y \\ 0 & 0 & 1 \end{bmatrix}$$

f is the focal length, and c_x, c_y is the principal point. s is the shear angle between the axes, and a is the aspect ratio. Multiplying a homogenous coordinate in 3d-space will lead to a shift by the principal point, and scaling by f upon perspective division.

The aim of Camera Calibration is to recover the intrinsics and/or extrinsics. If we denote the 3D points as $\mathbf{X}$, and the corresponding 2D points as $\mathbf{x}$, we can formulate the following set of equations:

$$\mathbf{x} = \lambda \begin{bmatrix} u \\ v \\ 1 \end{bmatrix}, \mathbf{P} = \begin{bmatrix} \mathbf{P_1^T} \\ \mathbf{P_2^T} \\ \mathbf{P_3^T} \end{bmatrix}$$

where u, v are the coordinates of the 2D point, and $\mathbf{P_i^T}$ are transposes of the columns of $\mathbf{P}$.

$$\mathbf{x} = \mathbf{P}\mathbf{X}$$

$$\lambda \begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = \begin{bmatrix} \mathbf{P_1^T} \\ \mathbf{P_2^T} \\ \mathbf{P_3^T} \end{bmatrix} \mathbf{X}$$

solve for λ, and we can write the above as:

$$\mathbf{P_3^T}\mathbf{X}u = \mathbf{P_1^T}\mathbf{X}$$
$$\mathbf{P_3^T}\mathbf{X}v = \mathbf{P_2^T}\mathbf{X}$$

Writing this system of equations as a postmultiplication of the $\mathbf{P_i^T}$'s gives us,

$$\mathbf{0} = \underbrace{\begin{bmatrix} \mathbf{X^T} & 0 & -\mathbf{X^T}u \\ 0 & \mathbf{X^T} & -\mathbf{X^T}v \end{bmatrix}}_{\mathbf{A}} \begin{bmatrix} \mathbf{P_1} \\ \mathbf{P_2} \\ \mathbf{P_3} \end{bmatrix}$$

If we solve the above equation for more than 6 points, we get more than 12 constraints (each equation contributes 2 constraints). The decomposition to intrinsic and extrinsic components can be acquired by an RQ decomposition, as the rotation $\mathbf{R}$ is an orthonormal matrix.

Distortions due to actual lens shapes can be modeled as a simple radial distance-based displacement of the 2d points, as follows:

$$\bar{\mathbf{x}} = [u\ v]^T(1 + k_1r^2 + k_2r^2 + ...)$$

For accurate SfM, it is crucial to get a good estimation of the camera projection matrix.

2.1.2 Feature Extraction

Features are descriptions of special locations of the image that are invariant under transformations of the image, spatial or otherwise.

Features can either be learned, or handcrafted. We outline the fundamentals of handcrafted features, as only these are potentially relevant in our pipeline to construct pointclouds.

Keypoint Extraction

We first have to identify interesting points in the image before we can invariantly describe them. One standard keypoint detector is the Harris detector, upon which most other handcrafted detectors are based. We outline below the central ideas of the Harris detector.

Without loss of generality, assume that a grayscale image I is used. For a small shift of $[\Delta x\ \Delta y]^T$ of some pixel located at $[x\ y]^T$ in a window W of the image I:

$$f(\Delta x, \Delta y) = \sum_{(x_k, y_k) \in W} (I(x_k, y_k) - I(x_k + \Delta x, y_k + \Delta y))^2$$

is the squared error sum obtained by shifting the window by $[\Delta x\ \Delta y]^T$.

We can approximate $I(x + \Delta x, y + \Delta y)$ by a Taylor expansion and keep only the first order terms. This leads us to the following expression for f:

$$f(\Delta x, \Delta y) = \sum_{(x,y) \in W} (I_x(x,y)\Delta x + I_y(x,y)\Delta y)^2$$

Expanding the squares, and rewriting as a quadratic matrix multiplication leads us to:

$$f(\Delta x, \Delta y) \approx [\Delta x \ \Delta y] \underbrace{\sum_{(x,y)\in W} \begin{bmatrix} I_x^2 & I_x I_y \\ I_x I_y & I_y^2 \end{bmatrix}}_{\mathrm{M}} [\Delta x \ \Delta y]^T$$

Here, M is called the structure tensor. We use the structure tensor to calculate the *Harris Response* as follows:

$$R = det(M) \ - \ k.trace(M)$$

We choose as keypoint only those locations $[x \ y]$ for which R is high. At these locations, both eigenvalues are high. This means that the error increases uniformly in all directions, and therefore the point is most likely a corner.

Descriptor extraction

There are several handcrafted feature descriptors which describe points of interest robustly, but SIFT (Shift Invariant Feature Transform) is the foundational one and the most widely used. The SIFT detector+descriptor works basically in the following stages:

SIFT Detector:

- Create scale pyramid of the image using a Gaussian kernel.
- Create a DoG (Difference of Gaussians) pyramid by taking the difference between successive scales.
- Detect extremal points by comparing to 8 neighbours in current, above and below scales.
- Filter out locations from previous step further with a Harris-like procedure described in the previous section.
- Assign as orientation of the point the dominant orientation of its gradient.

SIFT Descriptor: We know the scale, location and orientation of each keypoint.

- Around each keypoint, consider a 16x16 window. Subdivide this into 16 4x4 blocks.
- For each 4x4 block, create an 8-bin orientation histogram.
- Concatenate the 16 8-bin histograms to get a 128-dimensional vector. This is our final SIFT descriptor.

2.1.3 Feature Matching

Given descriptors for interest points for a pair of images (obtained perhaps with procedures like the ones outlined above), we seek to obtain corresponding points. This procedure is often done in a brute force manner for small images (each descriptor is compared to every other descriptor). However, this method is not feasible for images with a large number of features.

The standard solution to the above problem is to utilize tree methods, such as K-d trees, to speed up the search for a matching descriptor candidate. A popular method is the FLANN (Fast Approximate Nearest Neighbour search algorithm).

A pair of descriptors are compared using the L1 distance (for descriptors such as SIFT, SURF, etc) or the Hamming distance (for binary descriptors such as BRIEF).

The output of this stage in the pipeline is the set of corresponding points of the image pair.

2.1.4 Sparse Reconstruction

Triangulation

Given a set of corresponding points in a two-view setting, we can check for the location of the 3D point that projects on to these correspondences by intersecting rays from these correspondences. This idea is the basis of triangulating for the location of the 3d point.

The 3D point $\mathbf{X}$ projects on to the left image as $\mathbf{x_L} = \mathbf{P_L X}$ and onto the right image as $\mathbf{x_R} = \mathbf{P_R X}$, where $\mathbf{P_L}$ and $\mathbf{P_R}$ are the poses of the left and right images respectively. Since the camera poses and the 2D points are known, we can solve for the 3D point X in the above equations.

Refinement of reconstruction

The 3D points so constructed could have inaccuracies due to measurement errors and drift. A procedure called bundle adjustment is proposed to refine the triangulated points. Its objective function is:

$$\min \mathbf{a_j}, \mathbf{b_i} \sum_{i=1}^{n} \sum_{j=1}^{m} v_{ij} d(\mathbf{Q}(\mathbf{a_j}, \mathbf{b_i}), \mathbf{x_{ij}})^2$$

Camera i is parameterized by $\mathbf{a_i}$, and $\mathbf{b_j}$ is the j^{th} 3D point.$\mathbf{Q}(\mathbf{a_j}, \mathbf{b_i})$ is the predicted position of point i on image j, and $\mathbf{d}$ is the euclidean distance.

We can see that Bundle Adjustment minimizes overall reprojection error of each 3D point, and is optimized by the Levenberg-Marquardt algorithm.

2.2 The Basics of Convolutional Neural Networks (CNNs)

2.2.1 The Fundamentals of Deep Neural Networks

Neural networks are essentially a composition of alternating linear and nonlinear functions, nested to an arbitrary depth. These nonlinearities are usually fixed, and are called activation functions - eg. Logistic Sigmoid, tanh, Rectified Linear Unit (ReLU).

Logistic Sigmoid	$g(x) = \frac{1}{1+e^{-x}}$
tanh	$g(x) = \frac{e^{2x}-1}{e^{2x}+1}$
ReLU	$g(x) = max(0, x)$

The nesting level is called layer in neural network parlance.

The output at layer $l+1$ can be expressed as a function of the previous layer's output as follows:

$$a^{(l+1)} = g(\underbrace{W^{(l+1)}a^{(l)} + b^{(l+1)}}_{z^{(l)}})$$

This deeply composed function has enough degrees of freedom to theoretically fit any target function. The fitting procedure is performed by varying the weight matrix of all the layers ($W_{(l)}$). Gradient descent is the algorithm of choice to arrive at the optimal weights.

Gradient descent relies on varying the weights in the direction of the gradient of the loss function w.r.t the weights - this is the direction of steepest descent. However, to apply this procedure, we must first obtain the gradient of the loss function w.r.t the weights. This is obtained by a procedure christened backpropagation.

Backpropagation is actually a sequential application of the chain rule of calculus. If our error function is L, then backpropagating to find the weights of the first layer W_0 would function as follows:

$$\frac{\partial E}{\partial W^{(0)}} = \underbrace{\underbrace{\frac{\partial E}{\partial y^{(l)}}\frac{\partial y^{(l)}}{\partial z^{(l)}}}_{\delta^{(l)}}\frac{\partial z^{(l)}}{\partial y^{(l-1)}}\frac{\partial y^{(l-1)}}{\partial z^{(l-1)}}\cdots}_{\delta^{(l-1)}}\frac{\partial y^{(1)}}{\partial W_{(0)}}$$

The above architecture can technically learn to fit any function.

$f : \mathbb{R}^n \to \mathbb{R}^m$.

2.2.2 Convolutional Neural Networks

The standard feedforward neural network takes as input 1D vectors. Input of any spatial structure would have to first be unrolled into a 1D vector before it being used in this network. However, if we wanted to preserve spatial relationships during the learning procedure (eg. with images), we would need to formulate a new architecture.

Convolutional Neural Networks are the defacto method to address fitting functions to multi-dimensional data where spatial relationships are crucial. CNNs have been applied successfully to several computer vision tasks such as segmentation, classification, object recognition, etc. and high performance has been achieved in these tasks.

We begin by defining the application of the square shaped convolution operator H on image I of dimensions M X N as follows.

$$Conv(y,x) = \sum_{m=0}^{M} \sum_{n=0}^{N} H(m,n)I(y-m,x-n)$$

We can see that the convolution operation is a sum of the image intensities within the window, weighted by the values of the convolution kernel. For accesses outside the image area, padding (by reflection, zeros, etc.) is performed.

- The *Convolution Layer*, where the output is a set of images (called feature maps), that are the results of convolution of the input layer by a set of kernels of a certain dimension.

- The *Pooling/Subsampling Layer*, where a cluster of points around a each point is considered. The average or maximum of this cluster is taken as the output, thereby reducing the dimensionality of the feature maps.

- The *Fully Connected Layer* flattens the 2D feature maps into 1D vector outputs, which can then be used for conventional tasks such as classification or regression.

2.2.3 Semantic Segmentation

Semantic Segmentation is the task of predicting a class label for each pixel of a given image. The seminal work is the Fully Convolutional Network (FCN), which uses the features of a classification network. Experiments were performed on GoogLeNet ([Sze+15]), VGG ([SZ15]) and the AlexNet ([KSH12]) to predict a pixel-wise semantic mask for the image.

It is worth briefly looking into the details of GoogLeNet, VGG and AlexNet which are the basic classificaion networks and then proceed on to the usage of their features to predict the semantic mask via the CNN.

AlexNet

AlexNet is of historical significance to CNN architectures, as it was the first network to win the ImageNet classification challenge. It also introduced the architectural parlance that is now today's terminology of CNNs.

AlexNet is composed of 5 convolutional layers, 3 max-pooling layers, 2 normalization layers, 2 fully connected layers, and 1 softmax layer. Usually, the ReLU activation is used as the preferred nonlinearity. It addresses the problem of overfitting to data by using the principles of Dropout and data augmentation.

VGG

VGG is an AlexNet derivative, but made with the following improvements.

VGG, has several differences that separates it from similar models. Unlike AlexNet that uses large receptive fields, VGG uses small receptive fields (3x3 kernel with stride 1). The addition of 3 ReLU units makes the decision function more discriminative. There are also fewer parameters. VGG uses 1x1 convolutions to make the decision function behave in a more non-linear manner without changing the receptive fields. The smaller convolution filters allows VGG to have a larger number of weight layers, which leads to

improved performance. This, however, is a feature shared by GoogLeNet, which is delineated in the next section.

GoogLeNet

The main idea behind the GoogleNet architecture was the use of the Inception module, which created features of different receptive fields and aggregated them.

The GoogleNet has a depth of 22 layers, with 27 pooling layers. It consists of 9 inception modules stacked linearly in total, where the ends of the inception module is connected to a global average pooling layer. The intuition is that we learn features of varying receptive fields, each of which ”zoom-out” with a stacking of an inception module.

Fully Convolutional Networks for Semantic Segmentation

Fully Convolutional Neural Networks (FCNs) by [LSD15] in the use of CNNs for Semantic Segmentation.

FCNs start with a backbone classification network (such as AlexNet, VGG or GoogLeNet which we have described above), but with several adaptations which we will describe now .

In classification, conventionally, an input image is downsized and goes through the convolution layers and fully connected (FC) layers, and output one predicted label for the input image, as follows:

Conventional classification networks downsize the input image before sending them through the convolution and fully connected (FC) layers before predicting one label for the image. If we turn the FC layers into 1x1 convolutional layers and the image is not downsized, the output will not be a single label. The output is blown back to the input image resolution by upsampling.

Note that we can lose a lot of fine features while going through the pooling operations of the network, so the earlier features containing finer information are added onto the later stages. Depending on the stage at which we upsample and fuse features, we get FCN-32, FCN-16 and FCN-8's. Obviously, FCN-8's produce the best performance as it gets closest to the input resolution and incorporates fine and coarse feature fusions at several scales.

FCNs were built on by several authors incorporating more refinement stages ([BKC16]), efficient non-linear upsampling schemes ([LRB15]), adding global context [Zha+17] and pyramid pooling for context aggregation. [YK16] put forth dilated convolutions utilized in a context module to widen the receptive field, while [Val+17] submitted the idea of multiscale residual blocks with parallel dilated convolutions to enable quicker inference with no lessening of performance. [Che+17] initiated the use of multiple parallel dilated convolutions at different sampling rates for the purpose of multi-scale learning and then followed it with using CRFs for post-processing.

3 Related Work

3.1 Localization against a 3D Pointcloud

Traditionally, the task of Visual Localization is performed using the components of SfM. Given a query image whose pose we are required to find in some 3D reconstruction, the basic sequence of steps followed are:

- *Extract descriptors* for the query image.
- *Match* 2D query image keypoints to 3D Reconstruction points.
- *Refine pose* of the query image using the 2D-3D matches of the previous step, and the PnP algorithm.

Descriptor extraction has been detailed in the previous section. We will describe briefly the proceeding two steps.

3.1.1 2D-3D Matching

Once we have the features of the query image, we can associate them with corresponding image features in the reconstruction. Each image feature has also an association with some 3D point in the reconstruction, and by transitivity we can assert an association between the query keypoints and a 3D point of the reconstruction.

A tree structure like the k-d tree is used to hash the reconstruction's features of the reconstruction for rapid matching of the query image features against large reconstructions.

3.1.2 Pose Refinement with RANSAC + Perspective-n-Pose

Now that we know which 3D points are visible from the query image, we can try to figure out an affine transform of the 3D points that yields a minimum reprojection error when projected on the query image.

The reprojection error objective to be minimized is:

$$argmin_{R,t} \sum_{i=1}^{N} \|\hat{x}_i - x_i\|^2$$

The projection from 3D point to 2D point i is:

$$\mathbf{x_i} = \mathbf{K}[\mathbf{R}|\mathbf{t}]\mathbf{X_i}$$

The minimization is usually performed with the Levenberg-Marquardt algorithm.

To obtain a robust estimate of our pose, we sample sets of points and choose that pose which has maximum overall inliers (minimum overall reprojection error). This is the basic idea of the RANSAC procedure.

3.2 Localization with Global Image Descriptors

The basic idea of image based localization is to localize a query image by using strategies to find images (of known pose) in a database that are closest to it, and then compute a pose with respect to these set of ”close” images. The method of comparison employed between a pair of images here is the distance between their global image descriptors.While localizing against a huge pointcloud, the global descriptors can be used as a preliminary step in identifying a set of candidate images to form 2D-3D associations.

This reduces the search space considerably for matching query image features. The paper by [SLK12] is typical of these class of techniques.

3.2.1 Visual Vocabulary Trees

[NS06] is the seminal publication that exhibited early the possible use of Vocabulary trees to search visual cues. Intuitively speaking, visual words are a histogram over frequently occuring image content.This content is usually characterized by descriptors such as SIFT, etc. SIFT is the standard handcrafted descriptor for visual words, as they are highly distinctive.

- For a large, diverse database of images, aggregate all the SIFT descriptors from each image. This will be our training data.
- Run an initial k-means clustering on the training data and get the k cluster centers.
- The training dataset is then partitioned into k sets, where each set contains a cluster center and a set of descriptor vectors closest to it.
- Apply the previous two steps to each of the k sets, a predetermined number of steps. We now have a hierarchical k-means tree.

In the online phase, when we need to compute a visual word descriptor for an image, all we need to do is take every (SIFT) descriptor of the image and compute which of the k clusters of our vocabulary tree is closest to the word. This ”closeness” is computed by propagating the descriptor vector down the tree for each level-1 node, yielding a score for each node by computing inner products with each descriptor in the tree.

After doing this scoring for all the descriptors in the image and computing a histogram, we have our visual words description for the image.

3.2.2 VLAD

The VLAD descriptor, by [AZ13], serves as a global image descriptor. It also uses a vocabulary tree, and is computed as follows:

- Compute a visual vocabulary tree as described previously, for some database of images.
- Extract regions with an affine invariant detector.
- Describe regions with the 128 dimensional SIFT descriptor.
- Assign each descriptor to one of the k clusters of the vocabulary tree.
- Compute and accumulate residuals (difference between cluster center and assigned descriptors) for each of the k cluster centers. Formally, this can be defined as:

$$V(i,j) = \sum_{i=0}^{N} a_k((x)_i)(x_i(j) - c_k(j))$$

  where $x_i(j)$ and $c_k(j)$ are the i-th descriptor and k-th cluster center respectively. $a_k((x)_i)$ denotes the membership of the descriptor in cluster center k.

- Concatenate the final residuals from the k cluster centers into a $kx128$ dimensional dimensional descriptor, referred to as unnormalized VLAD.

The VLAD descriptor can then be normalized with an L2 function, a signed squared root, or the intra-normalization (wherein the VLAD is normalized within each cluster before concatenation, before L2 normalization).

3.2.3 NetVLAD

NetVLAD, by [Ara+15], seeks to incorporate the VLAD idea into a CNN, which can potentially learn much better features than the handcrafted SIFT descriptor. However, the obstacle to including VLAD as a layer would be the hard assignment of a descriptor to one of the clusters , i.e. $a_k(\mathbf{x})$. This is not differentiable. To get around this problem, we change the hard assignment to a softer assignment as follows:

$$a_k(\mathbf{x_i}) = \frac{e^{-\alpha\|\mathbf{x_i}-\mathbf{c_k}\|}}{\sum_{k'} e^{-\alpha\|\mathbf{x_i}-\mathbf{c'_k}\|}}$$

which assigns descriptor $\mathbf{x_i}$ to to cluster $\mathbf{c_k}$ depending on proximity. Expanding squares and cancelling terms, we can rewrite the assignment function as:

$$a_k(\mathbf{x_i}) = \frac{e^{\mathbf{w_k^T x_k}+b_k}}{\sum_{k'} e^{\mathbf{w'^T_k x'_k}+b'_k}}$$

where $\mathbf{w_k} = w\alpha\mathbf{c_k}$ and $b_k = -\alpha\|\mathbf{c_k}\|^2$. Implementation-wise, this soft-assignment function can be broken up as a convolution for the linear part, and then a softmax for the normalization.

3.3 Localization by Regressing Pose with CNNs

CNNs trained on image-pose pairs of a scene have enabled the of poses directly from a query image of that scene. The seminal work on this type of localization is PoseNet.

3.3.1 PoseNet

PoseNet takes as input a 224x224 RGB image, and regresses the 6-DoF pose of that image relative to the scene it was trained on. The output of the PoseNet is a 6 vector $[\mathbf{x}, \mathbf{q}]$, where $\mathbf{x}$ is the translation and $\mathbf{q}$ is a unit quaternion parameterizing rotation.

PoseNet Loss

The loss function for the CNN of PoseNet is defined as follows:

$$loss(I) = \|\hat{\mathbf{x}} - \mathbf{x}\|_2 + \beta \|\mathbf{q} - \frac{\mathbf{q}}{\|\mathbf{q}\|}\|_2$$

Note that the β parameter balances the relative importance of translation and rotation loss.

PoseNet Architecture

PoseNet, by [KGC15] utilizes as backbone the GoogleNet Architecture. GoogleNet is a 22 layer deep CNN which was state of the art at the time for classification.

However, the following modifications were made by the authors of PoseNet to convert the GoogLeNet backbone into a pose regressor network.

- The 3 softmax classifiers were replaced with an affine regressor (fully connected layers) which output a 7-dimension vector representing pose.
- Another fully connected layer of size 2048 was added before the final layer, so it could be used as a feature vector for localization.
- Normalize quaternion orientation vector to unit length at test time.

PoseNet was trained on image, pose pairs obtained from SfM and was able to get competitive results on standard datasets such as Cambridge ([KGC15]) and 7Scenes ([Sho+13]). It spawned several variants for visual localization that used its base architecture as a backbone, and we shall explain how we use one such variant (Mapnet) in our approach to solve the localization problem, as detailed in the next chapter.

3.3.2 VLocNet

An improvement upon PoseNet was VLocNet by [VRB18]. In order to estimate the global pose of a pair of query frames accurately with respect to some scene, cues from Visual Odometry (relative motion between frames) are used jointly with the global pose of the frames. The idea is that the relative motion estimates from VO would help constrict the search space considerably.

Given a pair of frames (I_t, I_{t-1}), the network aims to regress the absolute pose $\mathbf{p_t} = (\mathbf{x_t}, \mathbf{q_t})$ and the relative pose $\mathbf{p_{t,t-1}} = (\mathbf{x_{t,t-1}}, \mathbf{q_{t,t-1}})$ where $\mathbf{x}$ is the translation 3-vector, and $\mathbf{q}$ is the rotation 4-vector. The semantic branch predicts M_t, the pixelwise semantic mask of C classes for image I_t.

Architecture

The backbone network for the VLocNet is a modified ResNet-50 network [He+15].

However, the ResNet50 is modified as follows:

- Replace conventional ReLUs with ELU activation functions, which have been found to be more robust to noise and are faster to converge.
- After the fifth layer, add a global average pooling layer. Add three inner product layers (*fc1*, *fc2*, *fc3*) after, of dimensions 1024, 3, and 4. *fc2* and *fc3* regress the translation and rotation respectively.

Instead of directly regressing the pose from the network, an following extra step is introduced: the fusion of the previous timestep's final downsampling layer output with that of the current timestep's final downsampling layer output. This happens for pairs of frames in a sequence, and forces the network to learn motion cues in the sequence.

This scheme in combination with the *Geometric consistency loss* described in the following section enables the network to learn motion-specific cues in the temporal dimension.

Learning Pose Regression

The usual euclidean loss between predicted and ground truth pose is augmented here with an additional term to constrain the error between the relative motion predicted by the odometry stream of the network, and that available from the ground truth. If we denote our neural network as f, and its parameters as θ, we can build up to the final loss function as follows:

The translational and rotational losses between two consecutive frames are defined as:

$$\mathcal{L}_{xRel}(f(\theta|I_t)) = \|\mathbf{x}_{t,t-1} - (\hat{\mathbf{x}}_t - \hat{\mathbf{x}}_{t-1})\|$$

$$\mathcal{L}_{qRel}(f(\theta|I_t)) = \|\mathbf{q}_{t,t-1} - (\hat{\mathbf{q}}_t - \hat{\mathbf{q}}_{t-1})\|$$

These two losses are each weighted exponentially with a learnable exponent, and added together to form the relative loss.

$$\mathcal{L}_{Rel}(f(\theta|I_t)) = \mathcal{L}_{xRel}(f(\theta|I_t))exp(-\hat{s}_{x_{Rel}}) + \hat{s}_{x_{Rel}} + \mathcal{L}_{qRel}(f(\theta|I_t))exp(-\hat{s}_{q_{Rel}}) + \hat{s}_{q_{Rel}}$$

The absolute loss between pose prediction and ground truth has a similar structure:

$$\mathcal{L}_x(f(\theta|I_t)) = \|\mathbf{x}_t - \hat{\mathbf{x}}_t\|$$

$$\mathcal{L}_q(f(\theta|I_t)) = \|\mathbf{q}_t - \hat{\mathbf{q}}_t\|$$

The individual euclidean translation and rotation losses are put together in a cumulative absolute pose loss terms as follows:

$$\mathcal{L}_{Euc}(f(\theta|I_t)) = \mathcal{L}_x(f(\theta|I_t))exp(-\hat{s}_x) + \hat{s}_x + \mathcal{L}_q(f(\theta|I_t))exp(-\hat{s}_q) + \hat{s}_q$$

We put both relative and absolute losses together to get the following loss term:

$$\mathcal{L}_{Loc}(f(\theta|I_t)) = \mathcal{L}_{Rel}(f(\theta|I_t)) + \mathcal{L}_{Euc}(f(\theta|I_t))$$

Learning Visual Odometry

Another branch of the overall network learns to predict visual odometry $\mathbf{p}_{t,t-1} = (\mathbf{x}_{t,t-1}, \mathbf{q}_{t,t-1})$, given a pair of frames in a sequence (I_t, I_{t-1}). This is done by employing a dual stream architecture, wherein each branch is identical to one another, and is based on the ResNet-50 architecture.

The feature maps before the last downsampling stage of both stages are concatenated, convolved through the last residual block, followed by an inner product layer and two pose regressors for estimating pose of both frames. This loss between predicted and ground truth motion is expressed by the following loss function:

$$\mathcal{L}_{vo}(f(\theta|(I_t, I_{t-1}))) = \mathcal{L}_x(f(\theta|(I_t, I_{t-1})))exp(-\hat{s}_{x_{vo}}) + \hat{s}_{x_{vo}} + \mathcal{L}_q(f(\theta|(I_t, I_{t-1})))exp(-\hat{s}_{q_{vo}}) + \hat{s}_{q_{vo}}$$

The parameters between the odometry network here, and the global pose regression network are shared. This sharing enables an inductive transfer of information between both networks.

Learning Semantics

The authors propose two variants of a semantic learning branch: a single task architecture that predicts a pixel-wise segmentation mask for a monocular image, and a multitask architecture that incorporates self-supervised warping and adaptive fusion layers in the segmentation process.

For the *single-task architecture*, an encoder-decoder model is used. The encoder is a ResNet-50 architecture, which learns highly discriminative semantic features 16 times downsampled at the output layer. The decoder consists of two deconvolution layers, a skip convolution from the encoder for fusing high-resolution semantic maps and upsampling to the input feature resolution. At the output of the network, we get classification scores per pixel. We get the probability of assigning a class to a pixel given an image, by applying softmax to the per pixel classfication scores. The loss function is therefore defined as the maximum log-sum of all points in this distribution.

If we have a set of training images, $\mathcal{T} = (I_n, M_n), n = 1, ...N$, where $I_n = u_r | r = 1, ...\rho$ is the set of training images consisting of ρ pixels, and $M_n = m_r^n | r = 1, ...\rho$ is the set of corresponding ground truth masks with a semantic label per pixel, $m_n^r = 1, ...C$.

Using the per-pixel classification scores s_j we can model the probability of a pixel being assigned a semantic class with the softmax function, as follows:

$$p_j(u_r, \theta | I_n) = \frac{exp(s_j(u_r, \theta)}{\sum_k^C exp(s_k(u_r, \theta)}$$

The optimal network parameters θ is estimated by minimizing the following loss function:

$$\mathcal{L}_{\int}\rceil\}(\mathcal{T}, \theta) = \sum_{n=1}^{N} \sum_{r=1}^{\rho} \sum_{j=1}^{C} \delta_{m_n^r, j} p_j(u_r, \theta | I_n)$$

for $(I_n, M_n) \in \mathcal{T}$.

The *multitask architecture* utilizes a self-supervised warping method, wherein the pose of the previous timestep's image as predicted by the odometry stream is utilized. A depth image of the previous timestep, D_t, is predicted using a separate network [May+16], and the previous image's feature maps (outputs from layers *Res4f* and *Res5c* of the previous timesteps - this is marked in Figurerw3.7) are now warped onto the current image's features.

Formally, if the projection function is π, we can warp on the previous timestep's pixels u_r onto the current timestep $\hat{u_r}$ with relative pose $p_{1,t-1}$

$$\hat{u_r} = \pi(T(p_{1,t-1})\pi^{-1}(u_r, D_t(u_r)))$$

This operation allows robustness to camera angle deviations, object scale and frame distortions. It is also a feature augmenter, and thereby enforces consistent learning of consistent semantics.

Summary

The final loss function is an accumulation of the semantic, odometry and absolute pose losses as defined above. The reasons behind training absolute pose, odometry and semantics can be articulated with two points: to enable the absolute pose regression network to encode geometric an semantic information while training, and to enable inductive transfer between domain specific information.

This is the first network wherein the results of Localization is consistently equal to or better than the SfM/Feature desriptor approach. For detailed analysis of the results please refer the paper.

3.4 Cross-View Localization

Cross-View Localization is the task of localizing a query image against a database of images captured from a different view than the typical query image. The most usual example is localizing a ground-view image against a database of aerial/satellite imagery.

The seminal work in this area submitted by [WJ15], where they fine-tuned AlexNet on ImageNet and Places datasets. They proved the efficacy of CNNs as a feature extractor for cross-view image association by creating a dataset of the Charleston San Francisco aresa as follows: aerial imagery for a predetermined 40 sq.km region was downloaded from Bing maps, and was associated by GPS tagging to ground level images obtained via various open-source image databases such as flickr and Google StreetView. The pre-trained AlexNet was fed these images, and the final layer output is taken as the corresponding feature vector.

Their experiments were able to prove the high discriminative nature of CNNs for this purpose, showing the performance of their CNN features against the state of the art method by Lin.et.al ([LBH13]) at the time (which sought to learn the relationship between cross-view image pairs with a Support Vector machine based method).

Other methods that utilize CNNs for Cross View localization were then spawned. These myriad methods each utilize a different backbone CNN architecture, and incorporated more information about the image to make the CNN output more distinctive features.

Workman et al. show that fine- tuning the aerial branch by minimizing the distance between aerial and ground images results in improved localization performance. Vo and Hays (Vo and Hays 2016) conducted thorough evaluations on the network suited best for cross-view localization: i.e. binary classification (image retrieval), Triplet or Siamese CNNs. Hu et.al stacked a NetVLAD layer upon a VGG Network to endow VGG features the viewpoint invariance that NetVLAD possesses.Liu and Li (2019) added per-pixel orientation information into their CNN so that the features learned are sensitive to pixel orientation as another discriminative feature.

We outline here a method that is state of the art in the Cross View Localization domain, and builds upon the idea of using a CNN to learn features.

3.4.1 Optimal Feature Transport for Cross-View Image Geo-Localization

[Shi+19]. deals with cross-view geo localization, the task of finding the location of a given ground-view image in a large satellite map. As we have detailed before, this paper too is based on the premise of using DCNNs to extract features that can be used for comparing a ground-view image against an aerial image database. There are however two insights that are incorporated by the authors:

- Spatial layout of the features is to be taken into account - i.e. their relative position with respect to other features in the image plane.
- Take into consideration that the ground and aerial images are, for the lack of a better word, of different ”domains”.

To tackle both of these problems, the direct approach of learning a transport matrix to transfer the ground-domain features to the aerial domain features is utilized.

Two CNN branches (VGG, [SZ15].) are used to learn the aerial and ground features, but the improvisation is in the inclusion of the new feature transport module. Previous approaches are similar upto the usage of two CNN branches to extract features for the aerial and ground images. However they use a feature aggregation such as pooling before minimizing the loss between features, which discards spatial layout information.

The feature transport module is actually a set of cost matrices that transform ground to aerial features. This does away with the lossy aggregation operation, and explicitly models the domain change.

Feature Transport

$\mathbf{f}^i(g) \in R^{h \times w}$ and $\mathbf{f}^i(a) \in R^{h \times w}$ indicate the ground and aerial feature maps, wherein i is the feature channel number an h and w indicate the width and height of the features.

Ideally, both the ground and aerial images must be used in the computation of the cost matrix $\mathbf{C}$. For purposes of efficiency, we postpone the inclusion of aerial images to the loss function and instead use a regression block to compute $\mathbf{C}$.

Our transport matrix $\mathbf{P}$, has to satisfy the following strictly convex loss condition:

$$\mathbf{P}^* = argmin_P \ < \mathbf{P}, \mathbf{C} >_F - \lambda h(\mathbf{P})$$

where h is the entropy regularization, and the norm is the Frobenius norm. To minimize this particular functional objective, the Sinkhorn algorithm is utilized. The Sinkhorn algorithm first applies an exponential kernel on $\mathbf{C}$ and then iteratively converts $\mathbf{C}$ to a doubly stochastic matrix (rows and columns add up to one). If row and column normalizations are denoted as:

$$\mathbf{C}' = exp(-\lambda C)$$

$$\mathcal{N}_{i,j}^{r} = \frac{c'_{i,j}}{\sum_{k=1}^{N} c'_{k,j}}$$

$$\mathcal{N}_{i,j}^{c} = \frac{c'_{i,j}}{\sum_{k=1}^{N} c'_{i,k}}$$

For the m-th iteration, the Sinkhorn operation is given as

$$\mathbf{S}_m(\mathbf{C}') = \mathcal{N}^r(\mathcal{N}^c \mathcal{S}_{m-1}(\mathbf{C}'))$$

So when the iterations converge, $\mathbf{P}^* = \mathbf{S}_m(\mathbf{C}')$.

It is to be noted that the Sinkhorn operation is differentiable. Thus, when the ground feature maps are transported using the $\mathbf{P}^*$ and the loss is backpropagated, the domain transfer is also learned.

$$\mathcal{L}_{triplet} = log(1 + e^{\gamma(d_{pos} - d_{neg})})$$

where d_{pos} and d_{neg} denote the l2 distance of all the positive and negative aerial features from the anchor ground feature. The network outperforms the state of the art at the time of its writing (CVM-Net, [Hu+18]).

3.4.2 Geolocalization with 2.5D Maps

The method submitted by [Arm+17] describes a method to refine a localization estimate given an initial GPS location, and a 2.5D Map of the surrounding location. A 2.5D Map are a set of buildings which are normal map tiles but where building heights are also supplied.

Since we know a coarse pose, we can render the 2.5D map from point of view, and align building facades with the facades of the render map. A CNN is used to predict the pose which best causes this alignment, is applied until the pose converges.

Semantic Segmentation

A fully convolutional network is used to semantically segment the camera images into semantic classes. Only classes that are relevant to the pipeline, i.e. building, ground and sky are considered. The vertical and horizontal edges of these classes are extracted to help disambiguate the pose when the segmented and 2.5D rendered building facades are aligned.

Predicting a direction for pose refinement

The initial 6-DoF estimate can be obtained from the GPS location. Assuming the camera is at a fixed height, the ground plane is discretized in 8 direction classes defined in the camera coordinate system, including a class indicating that the current orientation is correct.

Thus, given a semantic view and a rendering of the 2.5D map for the pose of that view, the network predicts the best direction class for the pose update. Three classes for the orientation direction is also defined as left, right or no rotation at all.

More specifically we have two CNNs that predict translation and orientation classes as follows:

$$d_t = CNN_t(R_F, R_{HE}, R_{VE}, R_{BG}, S_F, S_{HE}, S_{VE}, S_{BG})$$

$$d_o = CNN_o(R_F, R_{HE}, R_{VE}, R_{BG}, S_F, S_{HE}, S_{VE}, S_{BG})$$

$R_F, R_{HE}, R_{VE}, R_{BG}$ are binary maps for classes facade (F), horizontal edge (HE), vertical edge(VE) and background (B) class obtained by rendering the 2.5D map by the coarse pose estimate. $S_F, S_{HE}, S_{VE}, S_{BG}$ are the probability maps for the same.

Pose Estimation Algorithm

CNN_t and CNN_o are applied iteratively, with a pose update after each iteration. The CNNs here do not provide a magnitude, so a line search method is used to determine the magnitude. As a quality criterion, we use the following maximum likelihood measure:

$$s_p = \sum_{c \in \{F,VE,HG,BG\}} \sum_{i \in R_c} log S_c^i$$

where S_c^i is the probability at location i for class c from the semantic segmentation, and $i \in R_c$ is the set of locations that are set to 1 in the rendered binary mask R_c.

For both direction and orientation predictions, linear updates are performed and those predictions that maximize the likelihood criterion specified above are kept as the final update. The iterations are stopped when both networks indicate the class of no movement.

4 Methodology

Our approach to the Visual Localization problem is outlined broadly by these three steps. See fig 4.1:

- Upon a dataset of overlapping map tiles of an area, train a Pose Regression Network (MapNet, [Bra+17]) to predict a 2d translation from a reference map tile.
- Convert pointcloud of the urban scene into a representation with building contours as seen from the road. This is the query map tile.
- With the trained MapNet, Infer the location of the *query map tile* directly upon the set of map tiles on which MapNet was trained.

The approach can be characterized as a hybrid Pose Regression, wherein the query image is a map-tile like representation. We will expound upon each of the three stages, starting with the details of training our Pose Regression network.

As we have built our system on the Oxford Robotcar dataset, we assume the presence of the following sensors:

- 2D LIDAR sensors with a field of view of 270 degrees.
- Inertial Motion Unit (IMU) equipped with Gyroscope.
- Multiple cameras covering the entire field of view.

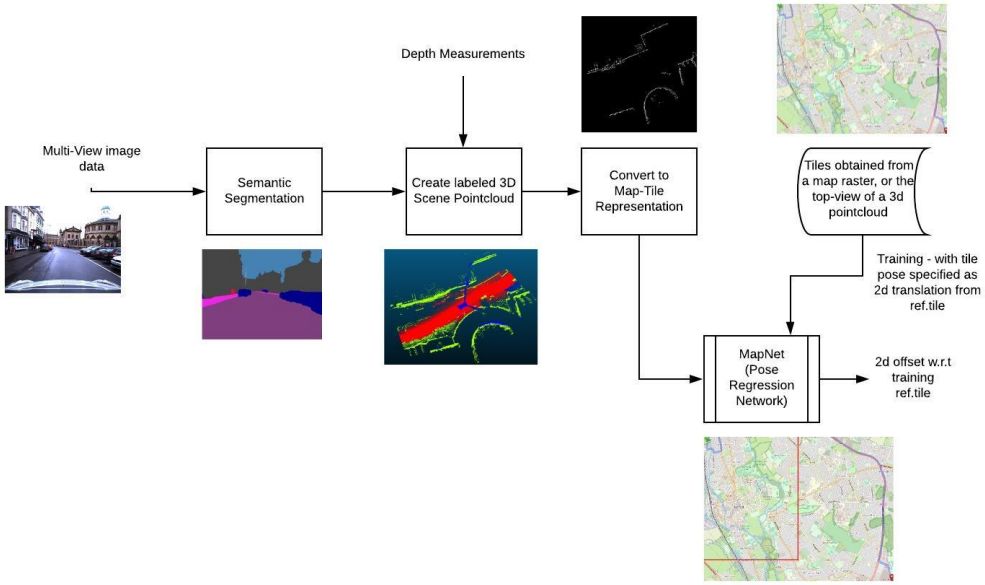


Figure 4.1: Our Pipeline: The input images are segmented, and their labels are transferred onto the pointcloud constructed from the scene. This labelled pointcloud is then converted into a map-tile like representation complete with OSM scale and orientation and is used as a query tile to infer the position of the vehicle directly on the map raster. [Mad+17], [Ope17]

4.1 Pose Regression

4.1.1 MapNet

MapNet ([Bra+17]), is a pose regresssion network that is based on the same idea as PoseNet([KGC15]) which was dealt with in the previous section. However, it mitigates the problem of PoseNet in that it reduces the noisy estimations by incorporating relative motion between frames as geometric constraints in the final loss function.

MapNet also has additional modules called MapNet+ which incorporates relative pose information from Visual Odometry in the loss function as a further constraint while training.

Pose Graph Optimization (PGO) is an additional component that allows MapNet to fuse incoming odometry information with the predictions from the network - this is done by enforcing the condition that pairwise relative poses from odometry must be positioned similar to the poses predicted by the network.

However, we will be using only the basic version of MapNet - that is, the version that enforces relative pose constraints between pairs of frame poses (as we get no visual odometry for map tiles).

The architecture of MapNet is basically a ResNet-34 modified as follows:

- Add global pooling layer after the last convolution layer.
- Add a fully convolutional layer of 2048 neurons with ReLU and dropout of 0.5.
- Finally, finish with a fully convolutional layer that regresses the 6-dof pose.

However, the 6-DoF pose that is regressed is a 6 parameter vector. This is due to the fact that the rotation is parametrized as the logarithm of the unit quaternion which results in a better performance than the standard 4 parameter quaternion.

The loss function is defined as follows:

$$\mathcal{L}_D(\Theta) = \Sigma_{i=1}^{|\mathcal{D}|} h(p_i, p_i^*)) + \Sigma_{i,j=1, i \neq j}^{|\mathcal{D}|} h(v_{ij}, v_{ij}^*)$$

The loss here penalizes any deviation from the absolute pose, and any deviation from the relative pose of the ground truth odometry. Here v_{ij} is the relative pose between frames i and j. The loss between poses p_i, $h(.)$, is defined as

$$h(p_i, p_i^*) = \|t - t^*\| e^{-\beta} + \beta + \|w - w^*\| e^{-\gamma} + \gamma$$

The β and γ terms balance the relative importance between translation (t) and rotation (w) loss terms respectively.

4.2 The Dataset

4.2.1 About the Dataset

The Dataset that we train MapNet on is drawn from OpenStreetMap ([Ope17]) depending on the city we want to localize in. We constructed a dataset for training the network, visualization from a section of the Oxford city area, see fig 4.2.

The GPS bounds are: (51.76868, -1.2730) to (51.73128, -1.192703), and it covers an area of approximately 25 sq.km.

The map is required to be of an urban area is chosen such that the number of buildings in the area is maximized, as building layout is after all the main discriminating feature for the entire pipeline.

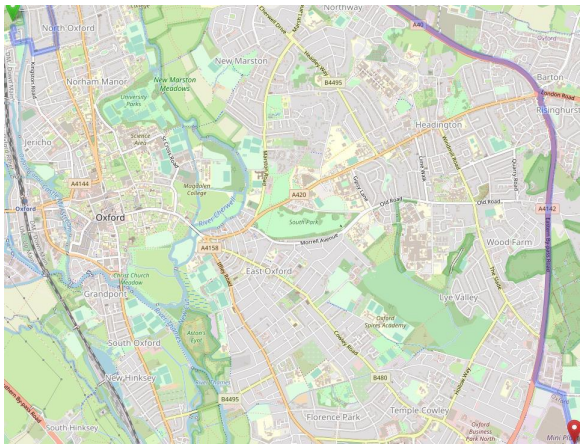


Figure 4.2: Training map for Oxford area. visualization from OpenStreetMap. [Ope17]

4.2.2 Obtaining the Map Tiles

The map, when obtained from OpenStreetMap is downloaded as a .mbtiles database file. One can use any Geographic Information System (GIS) software (we used QGIS), to select a rectangular area on this map containing the area upon which we want to localize and save this area as a set of tiles. In QGIS, this facility can be found under teh *Raster Tools* menu of the Toolbar.

We can choose to save the tiles at whatever zoom-level we want. For the purpose of our experiments, zoom-level 19 is the most suitable as it details approximately a 100 meter section of the road and buildings around it.

Map tiles obtained thus are saved in a folder, organized as images descending down the y-axis in a subfolder denoting the x-axis going to the right. These xy coordinates have a direct conversion to GPS coordinates, and back as follows:

```
def gps2xy(lat_deg, lon_deg, zoom):
    lat_rad = math.radians(lat_deg)
    n = 2.0 ** zoom
    xtile = float((lon_deg + 180.0) / 360.0 * n)
    ytile = float((1.0 - math.asinh(math.tan(lat_rad)) / math.pi) / 2.0 * n)
    return (xtile, ytile)
```

```
def xy2gps(xtile, ytile, zoom):
    n = 2.0 ** zoom
    lon_deg = xtile / n * 360.0 - 180.0
    lat_rad = math.atan(math.sinh(math.pi * (1 - 2 * ytile / n)))
    lat_deg = math.degrees(lat_rad)
    return (lat_deg, lon_deg)
```

4.2.3 Readying the Map Tiles for training

We cannot use the tiles obtained from OSM directly, as the query tiles we obtain will often contain incomplete information. This is directly due to the fact that the view from the car moving on a road does not contain the entire contour of the buildings around it, only a certain portion of the view (fig 4.3).

Figure 4.3: Incomplete view of buildings from a moving car. Note that only two facades of the building are visible. visualization from the Oxford Robotcar dataset, [Mad+17]

In the above figure we see that only two facades of the building are visible, which means that after processing the information from this scene we get an incomplete map tile. To make our system robust to this possibly incomplete information, we endeavour to make the *input map tiles resemble a rendering from the car point of view*.

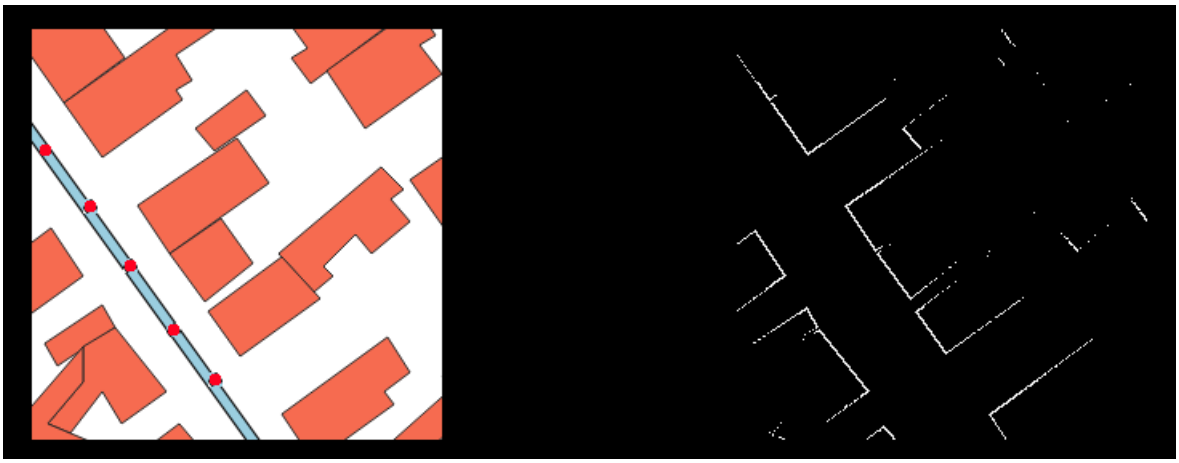


Figure 4.4: Left: road points sampled on a typical tile. Right: a rendering of the same tile as seen from the road points, [Mad+17]. On the left tile, buildings are colored orange and the road is blue. Red dots are points from where ray tests are made to create the right POV tile. [Ope17]

The typical map tile contains roads and buildings in separate colours, and thus we can sample points on the road of the tile. From these points, we can check which parts of the buildings are visible by checking where rays shooting out in all directions from the road points intersect the buildings. With this procedure we get the rendering on the right side of 4.4.

4.2.4 Assigning poses to tiles

MapNet assumes that the training image frames are presented in a sequence with poses associated with each frame. Conventionally, two successive frames fed to a pose regression network usually see the same scene but for a small forward motion. There is overlap between two successive frames which enables learning motion cues.

However, two map tiles ordinarily have no overlap. This we can rectify by creating new tiles positioned in between two successive tiles. The higher the number of new tiles we interpolate between any two given tiles, the better the dataset is for training features for a given trajectory.For our purposes,

we interpolate 4 new tiles between any two successive tiles positioned at increasing intervals of 0.25 of the tile width.

The sequence in which we present the tiles during training, starts from the top-left tile, moves to the right and then resumes again from the left once the row of tiles has run out. This, however, can by any continuous sequence of tiles.

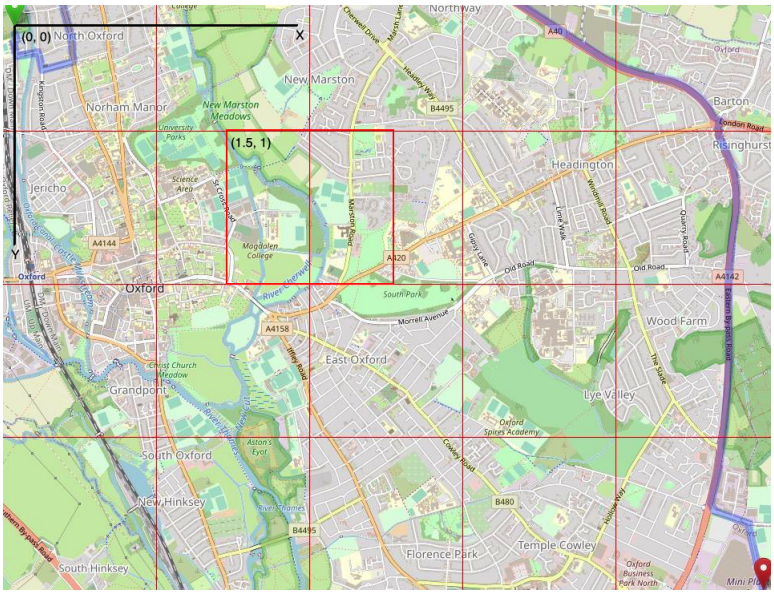


Figure 4.5: Choose the top-left tile as origin, and assign pose to every other tile. [Ope17]

We then choose the first tile in the trajectory as the origin, and assign the 2d offset from every tile to the pose of the first tile as the pose of that tile (fig 4.5). In the 6-dof scheme our tiles are not rotated, scaled or have any freedom in the z-axis. In other words, we only have reduced the 6-dof pose regression to 2-dof pose regression directly on the map raster.

4.3 Creating the Query Tile

We assume the scenario of a car driving through an urban area, with buildings visible on either side. Operating under this assumption, we adhere

to the following procedure to generate the query tile:

- Acquire pointcloud of the surrounding scene for a distance of approximately 100 meters of traversal.
- Semantically label pointcloud into buildings and roads.
- Project buildings onto the road plane. Sample points from the road plane and render building contours as seen from the road.
- Rotate this map-tile like representation to align with the true north, and scale it to the resolution of zoom-level 19 OSM tiles.

We will expound upon each of these steps now.

4.3.1 Acquisition of Pointcloud

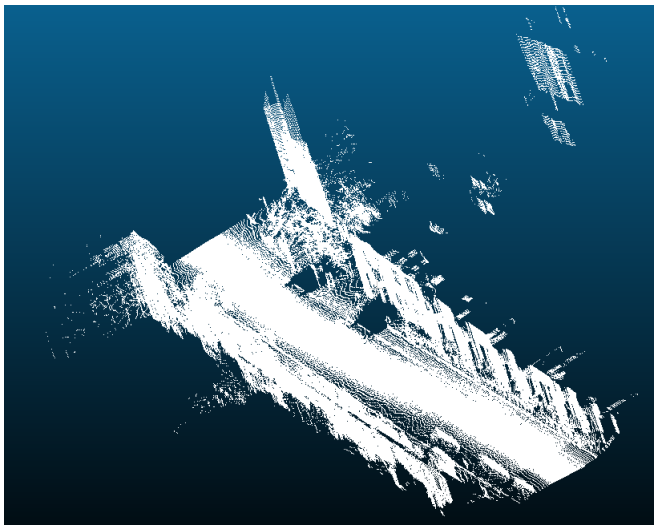


Figure 4.6: Typical LIDAR point cloud. Constructed from the Oxford Robotcar Dataset

There exist several ways in which we can acquire the pointcloud from a moving car. An inexpensive way to accomplish this would be to use cameras to capture a sequence of frames of the surrounding scene, and

exploit geometric constraints between successive frames to construct a 3D representation. This technique is called Visual Odometry, and its foundations lie in Structure from Motion which we have described in some detail in a previous section (2.1.1). [MMT15], [SSP19], and [Das18] are popular state of the art Visual Odometry methods applicable to our scenario to get pointclouds.

However, most Autonomous Driving setups can safely be assumed to possess a LIDAR sensor. The LIDAR sensor gives us depth measurements within a certain range (usually more than 50 meters) at any given time, and one can combine these depth measurements and the vehicle pose at the timestamp from the Inertial Measurement Unit to yield a pointcloud for a sequence of timestamps.

For the purpose of this thesis, we will assume the presence of LIDAR and IMU sensor data for the construction of the pointcloud (fig. 4.6).

4.3.2 Pointcloud Segmentation

We only require the points composing buildings and roads from the pointcloud to construct our map-tile representation of the scene. To accomplish this goal, we can use one of two techniques:

- Directly label the pointcloud using Pointcloud segmentation methods based on Deep Learning. Popular methods include [Qi+16] and [Aln+20].

- Project the pointcloud to segmented camera images with known poses, and infer labels thus.

We do not use the first approach of segmenting the pointcloud directly, as the network has to be retrained to cope with different kinds of urban scenes. The lack of generalization makes it hard to deploy quickly or reliably.

We use the second approach, opting to go with the MSeg method ([Lam+20]) to semantically segment images. In the ideal case, if the labels were transferred correctly onto the pointcloud we could move directly to the next step of converting it into the query map tile representation.

The intuition is that if we have constructed a pointcloud for a certain time period, then it can be projected to images taken in that time period, provided that their poses are known. We have described this projection procedure in detail in 2.1.1. We then decide the label of each pointcloud point by voting on which label it projects to on the set of images. More precisely, we outline the pseudocode for labeling below:

Algorithm 1 Algorithm to label pointcloud by projecting to images

Require: $S = \{S_1, S_2, ..S_N\}$, the set of segmented images for associated with the pointcloud.
$T = \{T_1, T_2, ...T_N\}$, the set of absolute poses corresponding to each image.
$P = \{p_1, p_2, ...p_M\}$, the set of 3d points in the pointcloud.

foreach p_k in P:
label-list: initialize an empty list for voting on a label for the 3d point p_k
foreach T_j in T:
Retrieve label l for p_k by projecting p_k onto labeled image S_j with pose T_j
. push label l onto the label-list
the label of p_k is the label that occurs most in label-list

The voting policy for labelling keypoints helps to mitigate the noise (fig. 4.7), but inaccuracies of the IMU will inevitably lead to mislabelings. We resort to a series of procedures which we will describe in the following section.

4.3.3 Conversion of the labelled pointcloud to map tile

Once we have labelled the pointcloud as building and road, we can fit a plane through the road points. For this we use ([ZPK18])'s plane segmentation method.

Once we have the road plane, we can project the building points onto it (fig. 4.8). The value of the projected pixel will be the "height" of the building point i.e. its distance from the road plane.

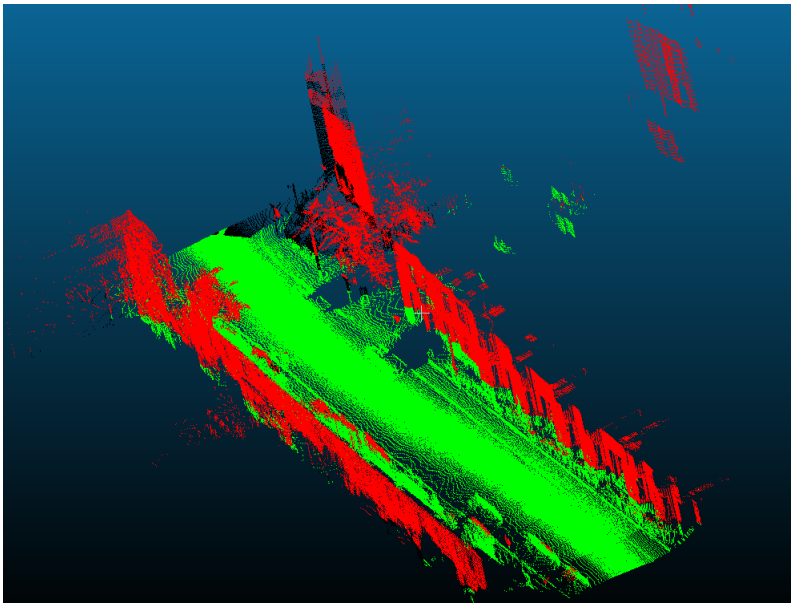


Figure 4.7: Segmentation of the pointcloud (Red: Building, Green: Road). Note noisy segmentation of trees as building.

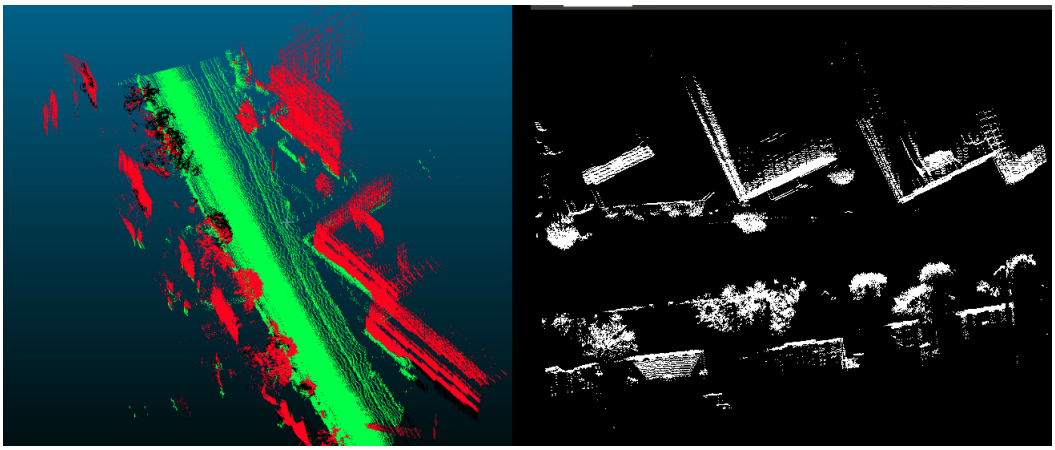


Figure 4.8: Left: Labelled pointcloud, Right: Projecting pointcloud on to the road plane

We then sample points from behind buildings on either sides of the road as in fig. 4.9, and do ray-intersection tests in all directions from these points to trace out building contours. This way, we avoid hitting tree and other random noise which we would encounter if we shot our sampling rays from somewhere on the road.

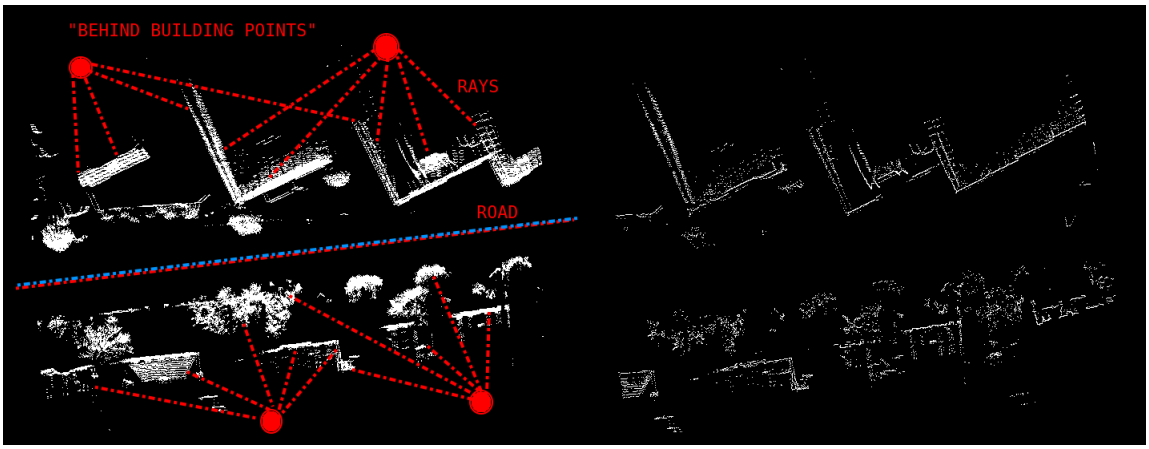


Figure 4.9: Left: Projected Pointcloud, Right: Sampling of projected pointcloud from points behind the buildings using omnidirectional line-intersections.

We then scan each column of this projected image, and keep only the 5 highest valued points, i.e. the 5 points of maximum height (fig 4.10). This strategy is useful to discard noisy tree points which are often lower than the tops of the buildings.

This strategy does lead to images where most noisy points mixed with building points are removed. However, several images have only trees on one side and no buildings at all or when trees are higher than buildings. In this case, our simple approach will keep the trees as it is only a way to filter noise assuming presence of buildings.

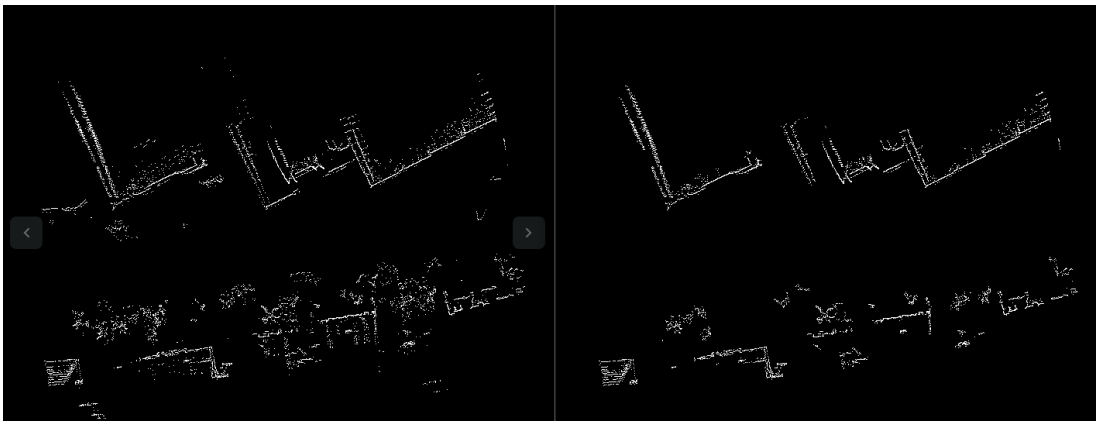


Figure 4.10: Left: Sampled projection, Right: Noise reduction of sampled projection by keeping highest pixels per column

4.3.4 Rotation to align with True North, and scaling to OSM tile resolution

The map-like representation that we get from the previous step is at an arbitrary resolution, and is not aligned with the OSM map tile direction. To be used for inference, this projected tile has to be aligned in the true north direction.

As we assume the presence of a gyroscope, we can get the attitude (orientation of the car with respect to the magnetic north) of the car for the duration of the trajectory for which we have constructed our query tile. To align with the *True North* which is the default orientation of the OSM tile, we must add the magnetic declineation of the car to the attitude. The magnetic declineation is dependent on the latitude, and can be queried from most standard GIS libraries. We use *geomap* for this purpose (https://github.com/amiralis/geo-mapper).

We must also make sure that our query-tile is at a resolution similar to OSM map tiles. We do this by calculating the extreme points of the road, and accumulate the distance travelled in meters for the map tile (with car measurements, or IMU). With this, we have an estimate of the pixel per meter resolution for our query tile. We will scale this so that it matches

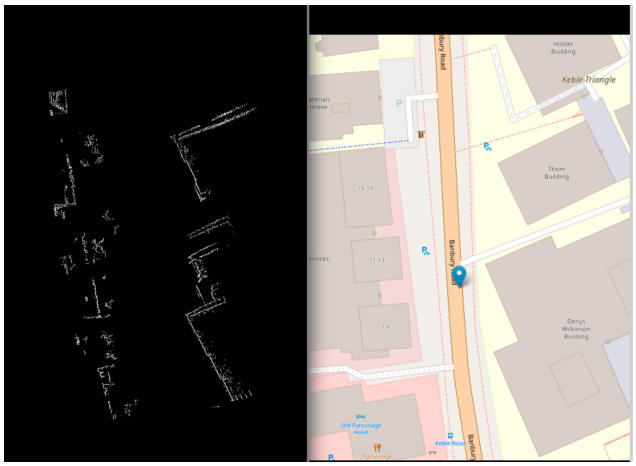


Figure 4.11: Left: Final Sampled Projection rotated to align with true north, Right: Corresponding map tile from OSM. Alignment is good. [Ope17]

OSM map tile resolution which is about 0.298 meters/pixel for the Oxford Dataset (fig. 4.11).

However, it must be noted that our scaling to OSM resolution is not precise as we cannot calculate the exact pixels travelled in case of a non-linear road. This may lead to noisy estimates during inference.

5 Evaluation

We've performed the following evaluations to understand the performance of our system:

- Evaluating network performance on training/test split of OSM Tiles - a check to see if the network is able to localize given query tiles from the dataset it has not seen before.

- Evaluating effects of rotating and scaling tiles - to show that under our current setup, the query maptile is sensitive to rotation and scaling.

- Evaluating the localization of several points of a drive on the Oxford RobotCar dataset - evaluating on a real scenario, wherein we construct our query map tiles from a real scene with the methods outlined in the previous section.

5.0.1 Evaluating network performance on training/test split of OSM Tiles

We begin with an evaluation of the MapNet network trained on map tiles obtained from the Oxford Area (fig 5.1). The GPS bounds delineating the area of our map tiles are the following:

(lat_min, lon_min) = (-1.272869110107422, 51.75041438844966)
(lat_max, lon_max) = (-1.2330436706542969, 51.76868973964186)

As described in our methodology section, we sample from this map area overlapping map tiles at zoom level 19. After converting these tiles into binary images which depict car point-of-view building contours. Our network

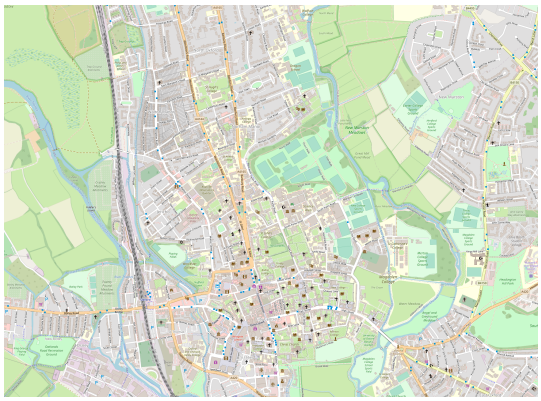


Figure 5.1: Area to train our localization on map tiles. visualization from OSM.

is trained on pairs of these tiles and their offset from a reference tile (approx. 16800 tiles).

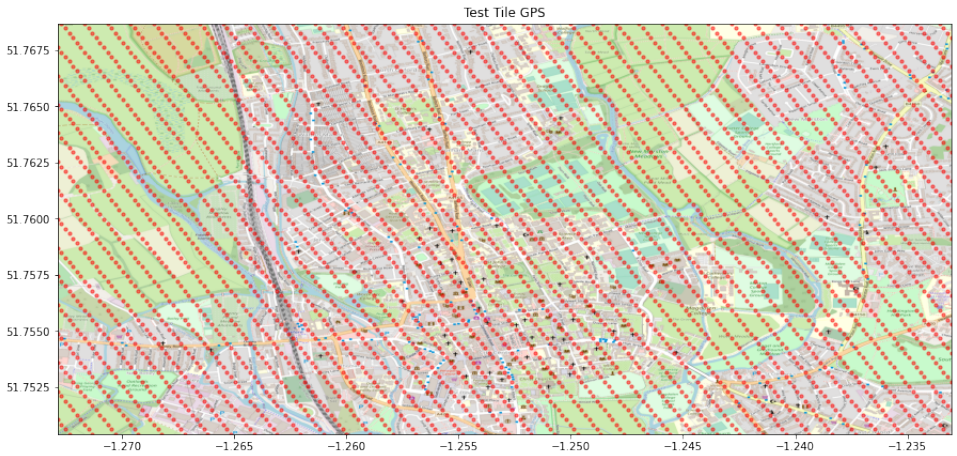


Figure 5.2: Red dots indicate locations where we sample tiles. For visibility purposes, we only show samples at fixed intervals. visualization from OSM. [Ope17]

We uniformly sample every 8th tile from our original dataset, and include it in our test dataset for evaluation containing approx. 2100 tiles (fig 5.2). Note that the test set follows the same trajectory as the training tile sequence. After training for 200 epochs, we predict the positions of our test tiles fig 5.3.

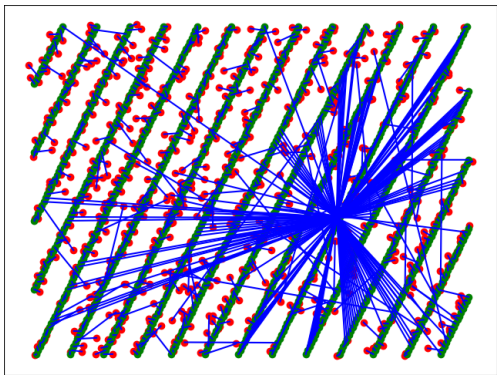


Figure 5.3: Predictions of MapNet on test tiles sampled from the training tile trajectory. Red Dots - Prediction, Green - Ground Truth. Blue lines - prediction to ground truth mappings

We note first that a majority of our predictions are close to the ground truth, with a few predictions being quite noisy (note that several noisy predictions tend toward almost the same (wrong) tile). Figure 5.4 showcases more precise statistics, i.e. a distribution of tiles over localization error in tile-widths.

As we can see, more than 75% of the test tiles lie within 5 error tile-widths of the ground truth tile, with more than 50% within 2 tile widths. Note that several high-noise predictions go to a region in the right side of our trajectory - this is actually a blank tile that the network seems to default to when it is unable to regress the query tile with a high confidence. The blank tile we refer to here is an empty tile in the map raster wherein no building or road exists.

Effects of Rotation, Translation and Scaling of query tile

While creating the query tile from the pointcloud of the scene, we take care to ensure that our query tile is oriented in the True North direction (which is the direction of the default orientation of OpenStreetMap tiles) and to ensure that the distance encoded by each pixel roughly matches the resolution of the respective OSM tile.

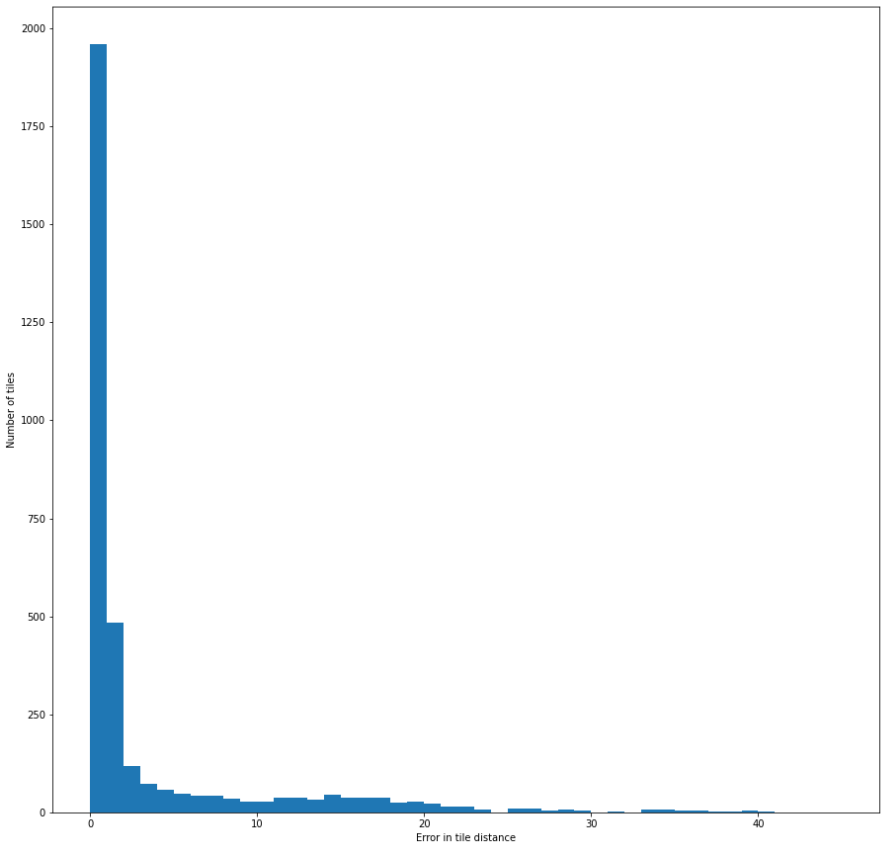


Figure 5.4: X: tile error widths, Y: no of tiles. Distribution of test tile localization error. These tiles' position are inferred without modification

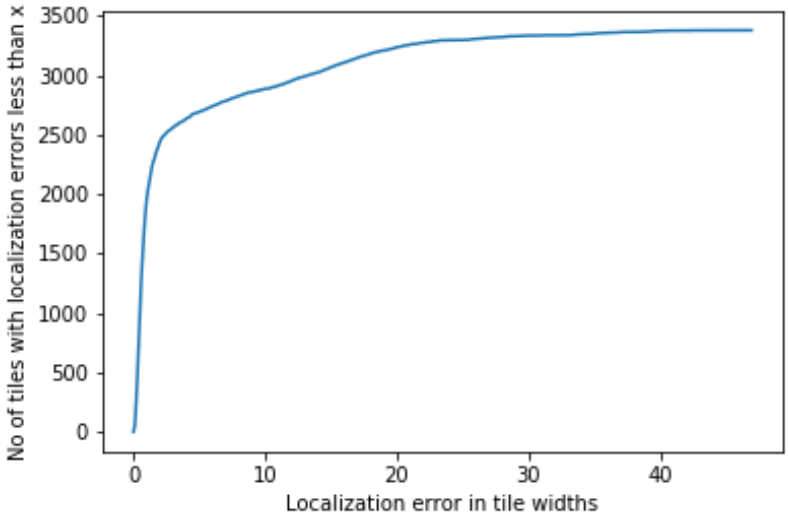


Figure 5.5: Plots the cumulative error distribution of test tiles over localization errors, a cumulative version of 5.4

We prove the necessity for this procedure by showing the changes in accuracy of localization when varying the orientation, scale and translation of the query tiles.

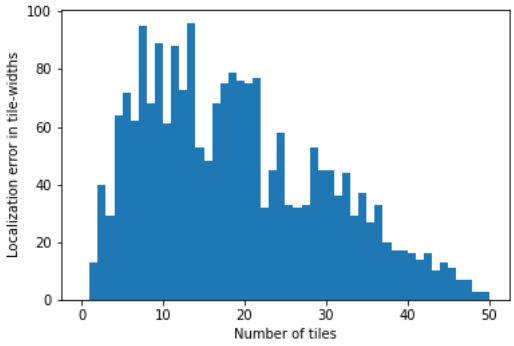
Rotation

We check the error profile for each tile, for orientations at an interval of 30 degrees between them. We note from the plots below that predictions get significantly noisier as we rotate the tiles away from their true orientation (fig 5.6).

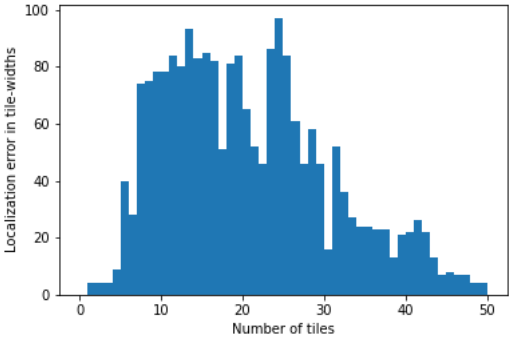
Scaling

Similarly, we vary the scale of the query tiles and check the prediction error statistics. The error plots below show clearly that the closer we get to the original scale, the lower the likelihood of an erroneous prediction (fig 5.7).

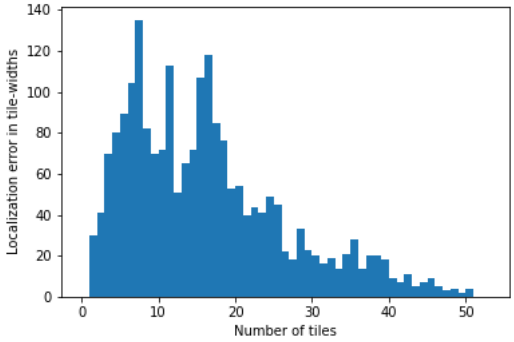
Effects of rotating test tiles at a 30 degree increment



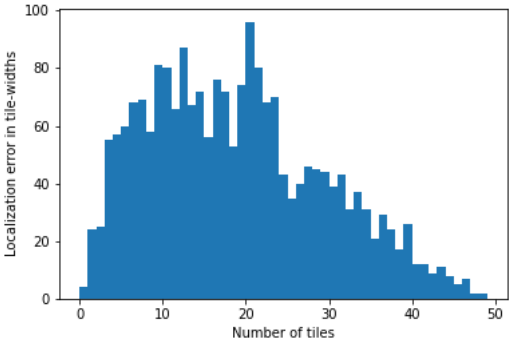
(a) 30 degrees error distribution



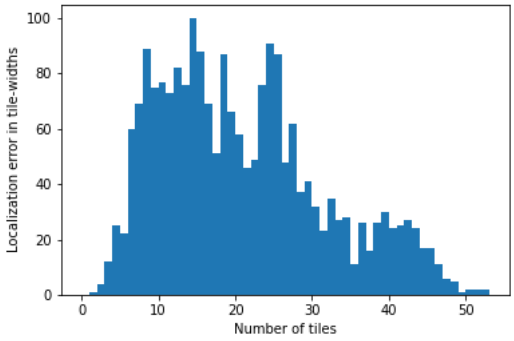
(b) 60 degrees error distribution



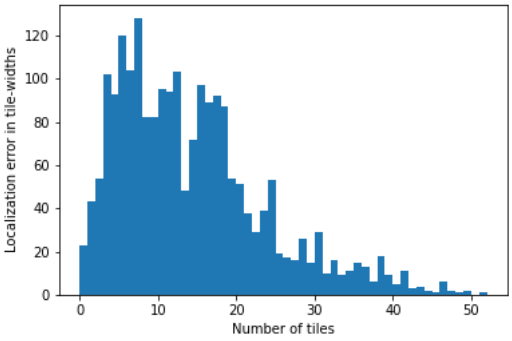
(c) 90 degrees error distribution



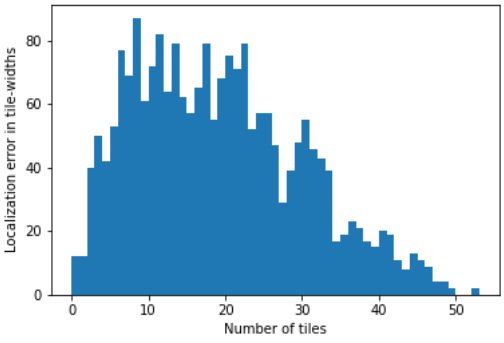
(d) 120 degrees error distribution



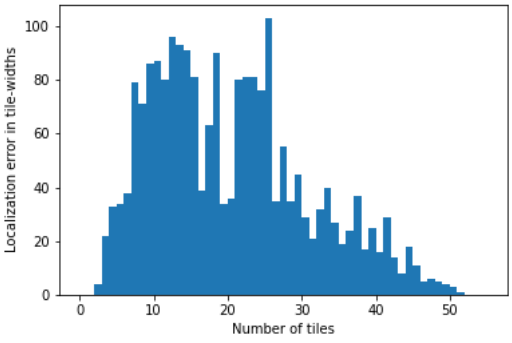
(e) 150 degrees



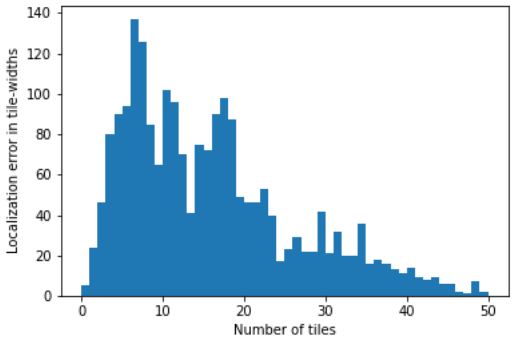
(f) 150 degrees error distribution



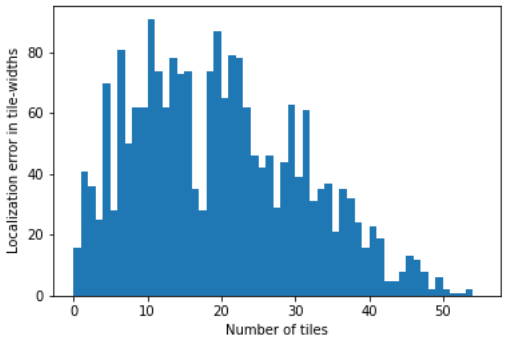
(g) 180 degrees



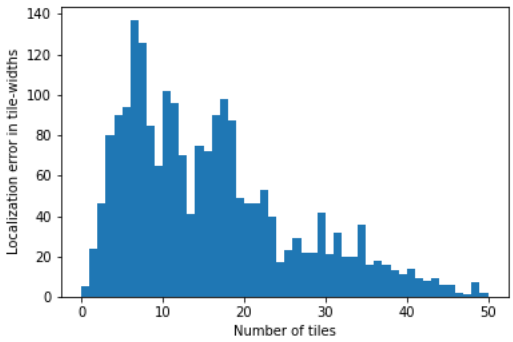
(h) 180 degrees error distribution



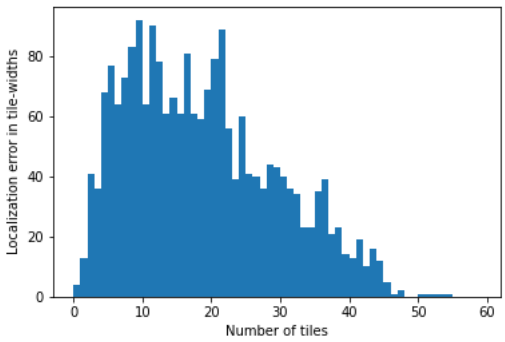
(a) 210 degrees



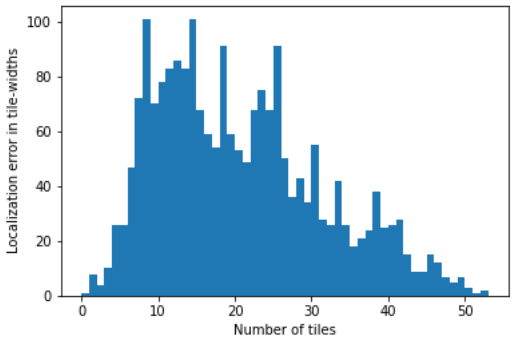
(b) 210 degrees error distribution



(c) 270 degrees error distribution



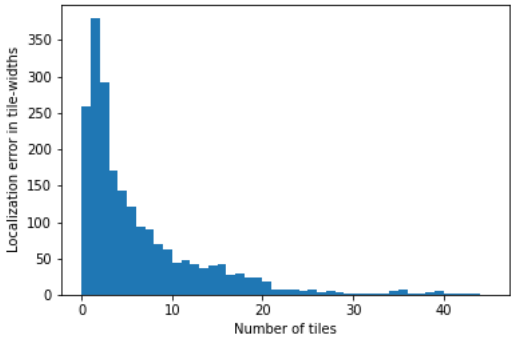
(d) 300 degrees



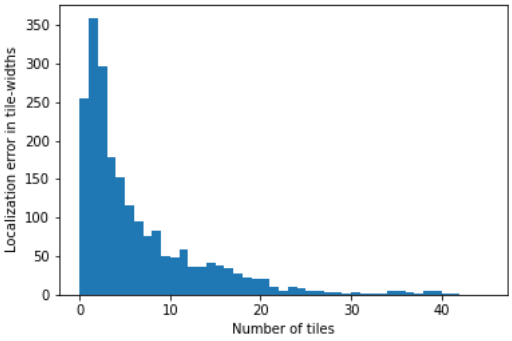
(e) 300 degrees error distribution

Figure 5.6: Effect of rotation on localization. We check the error for rotation steps of 30 degrees for the entire dataset, and show that as deviations from the actual orientation yields higher errors.

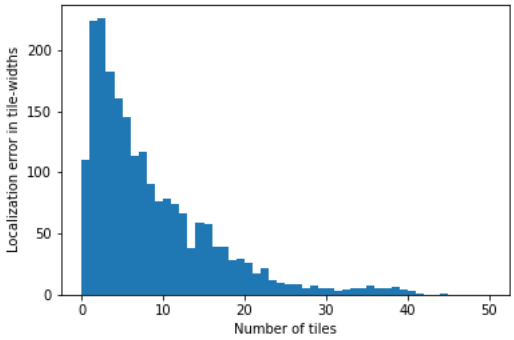
Effects of scaling the test tiles



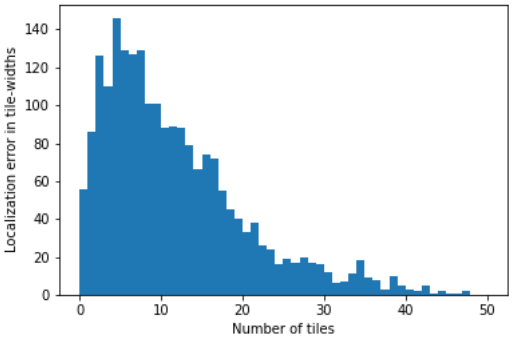
(a) 0.3 scaling error distribution



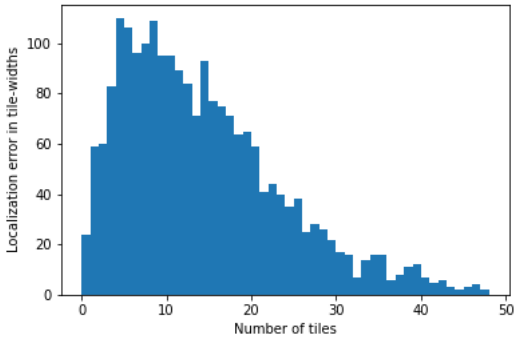
(b) 0.7 scalings error distribution



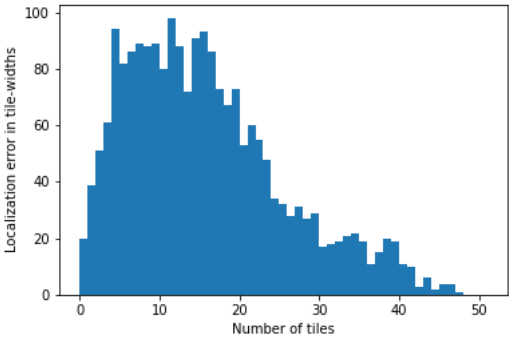
(c) 1.1 scaling error distribution



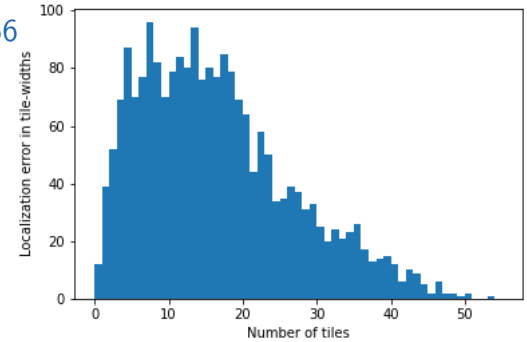
(d) 1.5 scaling error distribution



(e) 1.9 scaling error distribution



(f) 2.3 scaling error distribution



(g) 2.7 scaling error distribution

Figure 5.7: Effect of scaling on localization. We check the error for scaling steps of 0.4 scale increments for the entire dataset, and show that as deviations from the actual orientation yields higher errors.

5.0.2 Evaluation on query tiles constructed from the actual Oxford Sequence

As we have stated, we evaluate our method on the Oxford Robotcar Dataset (Maddern, Pascoe, Linegar, and Newman [Mad+17]). All the sequences of the RobotCar dataset are runs on a fixed route in different times and conditions. We choose the sequence **2015-03-17-11-08-44** for our evaluations (fig 5.8).

Evaluating for the Oxford RobotCar sequence

For evaluation, we choose those locations in our oxford sequence that are as close as possible to those tiles in our training sequence that have a low localization error (within 1 tile distance). This way, we can make a fair evaluation in the sense that the locations we are testing from the Oxford Dataset are known to have done well when represented with standard OSM tiles.

Even though our Oxford Robotcar query locations will not overlap exactly with their corresponding OSM tiles of the training dataset, we can still make an evaluation of localization based on how close we can get to our OSM tile representation.

We have sampled query locations almost uniformly from the oxford trajectory(fig 5.9). We manually choose those locations with distinct building geometry, while rejecting those that are empty of any structure.

We observe from the below figure that we have sampled locations uniformly from the original oxford driving trajectory.

As we have described in our methodology section, we create query map tiles at the locations shown above and evaluate how well the network localizes our query tiles at the locations sampled from the oxford robotcar trajectory

Figure 5.8: Trajectory of Oxford Dataset drive

((fig 5.10)). We also separately show the localizations at different errors (fig 5.11, fig 5.12. fig 5.13. fig 5.14, fig 5.15, fig 5.16)

We show below statistics of our system's performance:

We observe that the predictions are within 6 tile widths of the ground truth (fig 5.17). This error appears to be occuring in directions different than that of the driving trajectory. This is consistent with the behaviour of pose regression with a CNN, wherein the inferences are inherently noisy but drift-free. However, it has to be noted that we are able to narrow down the location to within a 5 tile radius - a search space of at most 55 tiles, given that the total search space is approximately 2500 tiles.

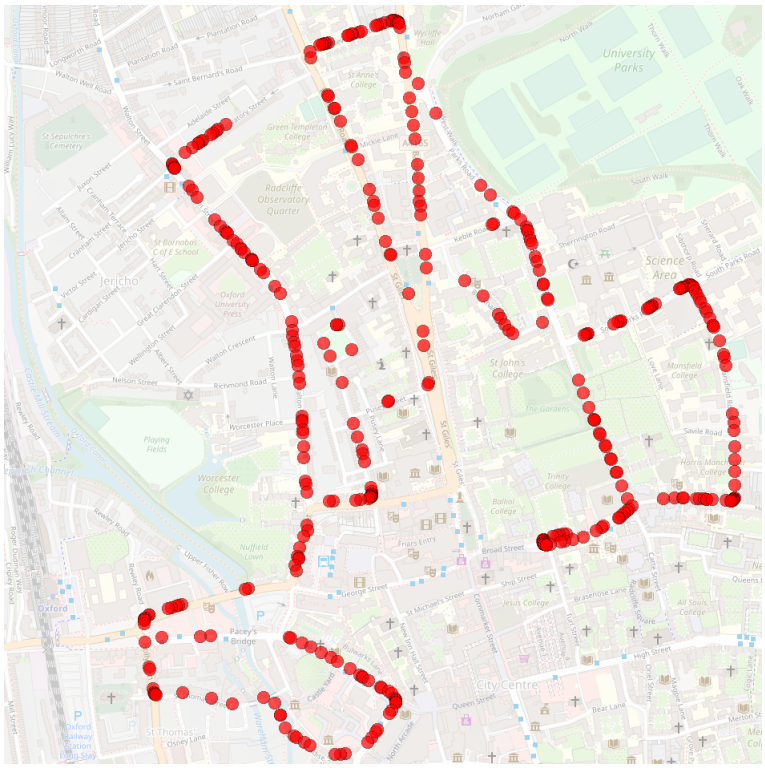


Figure 5.9: Query locations for evaluation

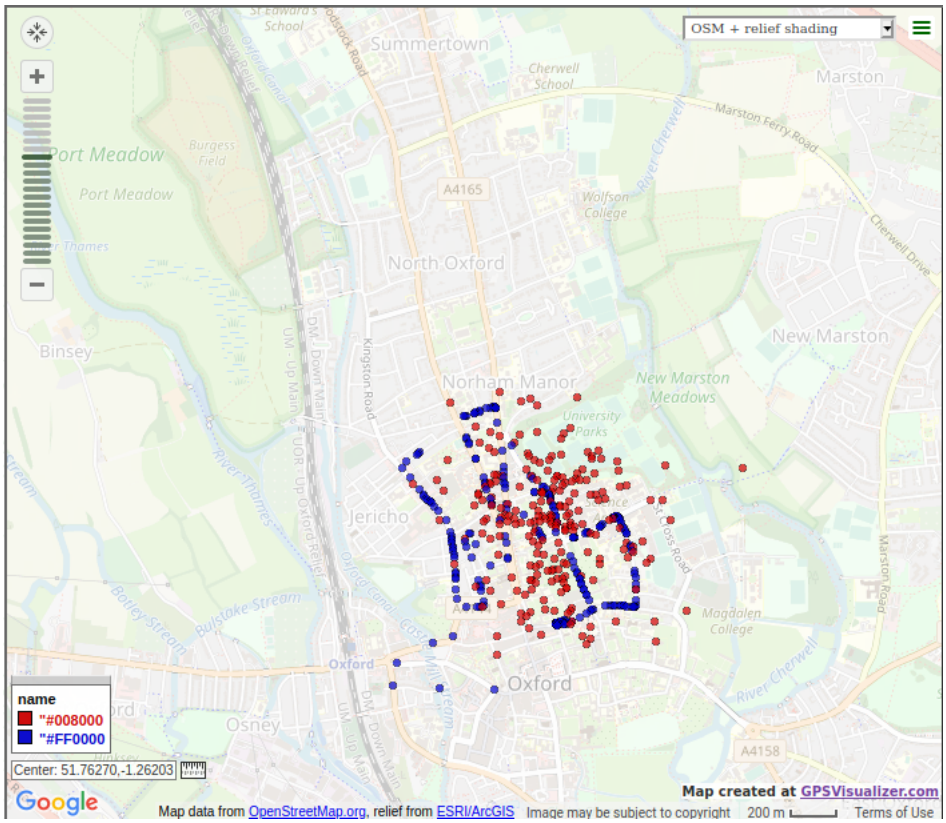


Figure 5.10: Blue - GPS positions of ground truth. Red - GPS positions of prediction

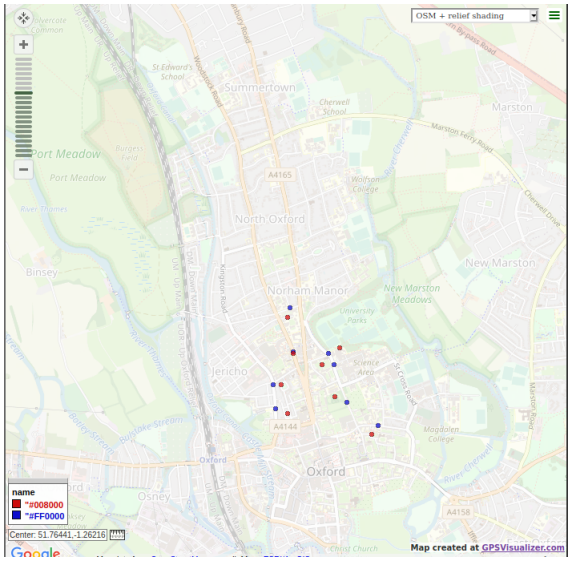


Figure 5.11: Blue - GPS positions of ground truth. Red - GPS positions of prediction. Locations with errors less than 1 Tile Width

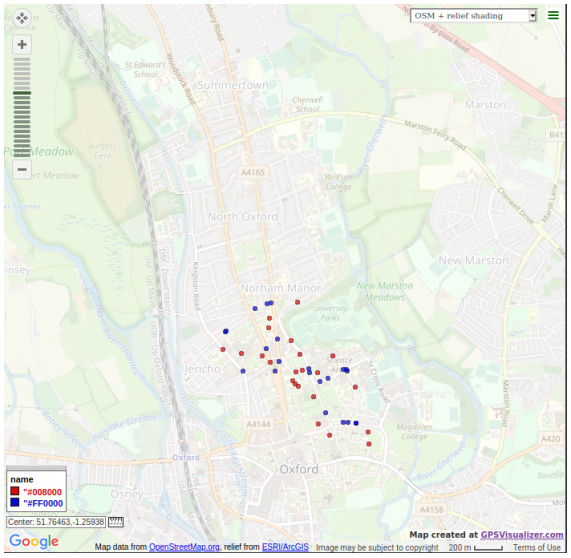


Figure 5.12: Blue - GPS positions of ground truth. Red - GPS positions of prediction.Locations with errors greater than 1 and less than 2 Tile Width

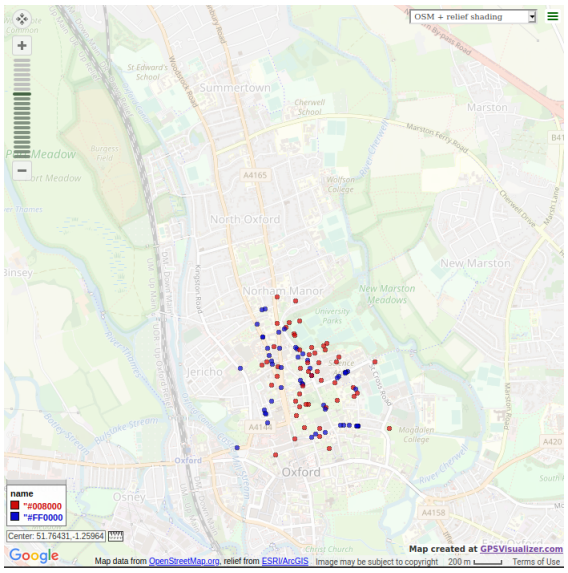


Figure 5.13: Blue - GPS positions of ground truth. Red - GPS positions of prediction. Locations with errors greater than 2 and less than 3 Tile Width. The map shown covers the whole search area.

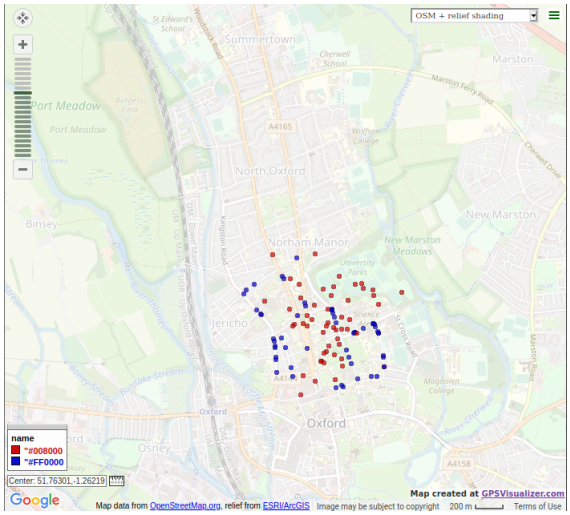


Figure 5.14: Blue - GPS positions of ground truth. Red - GPS positions of prediction. Locations with errors greater than 3 and less than 4 Tile Width. The map shown covers the whole search area.

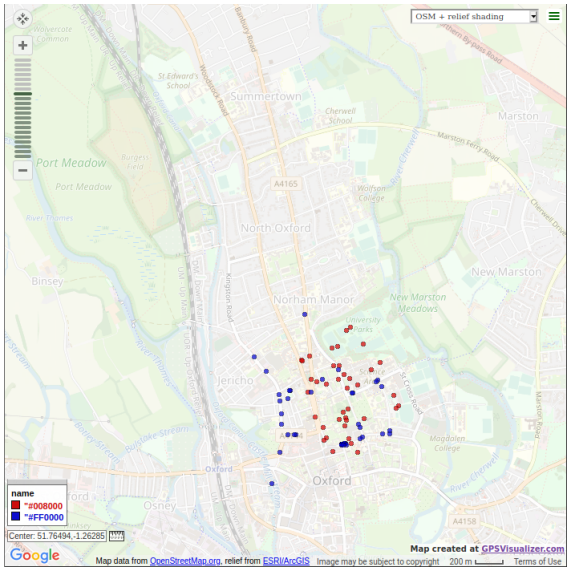


Figure 5.15: Blue - GPS positions of ground truth. Red - GPS positions of prediction. Locations with errors greater than 4 and less than 5 Tile Widths.

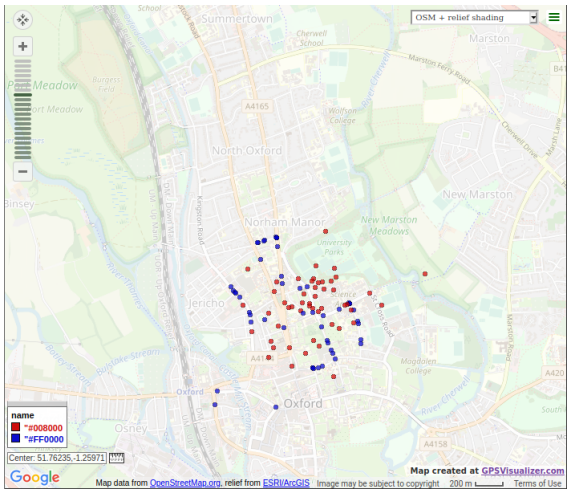


Figure 5.16: Blue - GPS positions of ground truth. Red - GPS positions of prediction. Locations with errors greater than 5 and less than 6 Tile Widths.

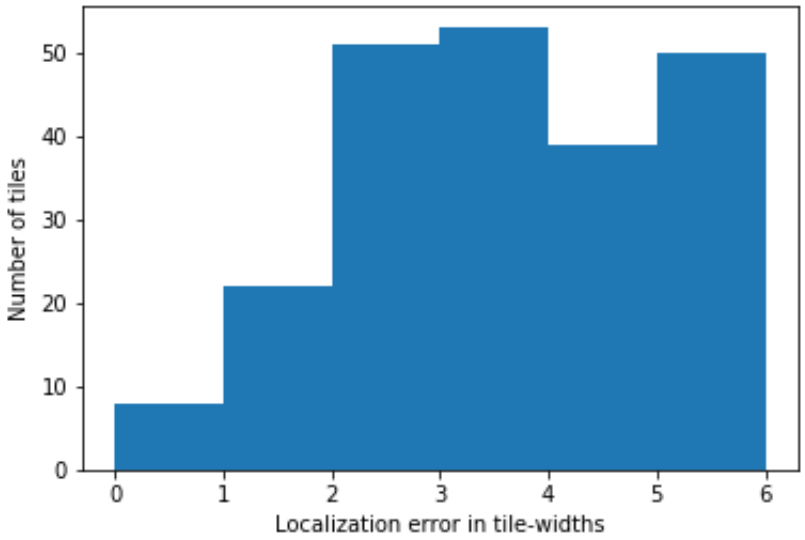


Figure 5.17: Localization error distribution over 238 locations in Oxford Trajectory. The map shown covers the whole search area.

A qualitative evaluation of the results

We display a set of (query tile, predicted map tile) pairs falling in different bands of localization error (in tile widths) to get a sense of how exactly our method performs. (fig 5.18, fig 5.19, fig 5.20, fig 5.21, fig 5.22, fig **??**)

Examples with Localization errors less than 1 Tile Width

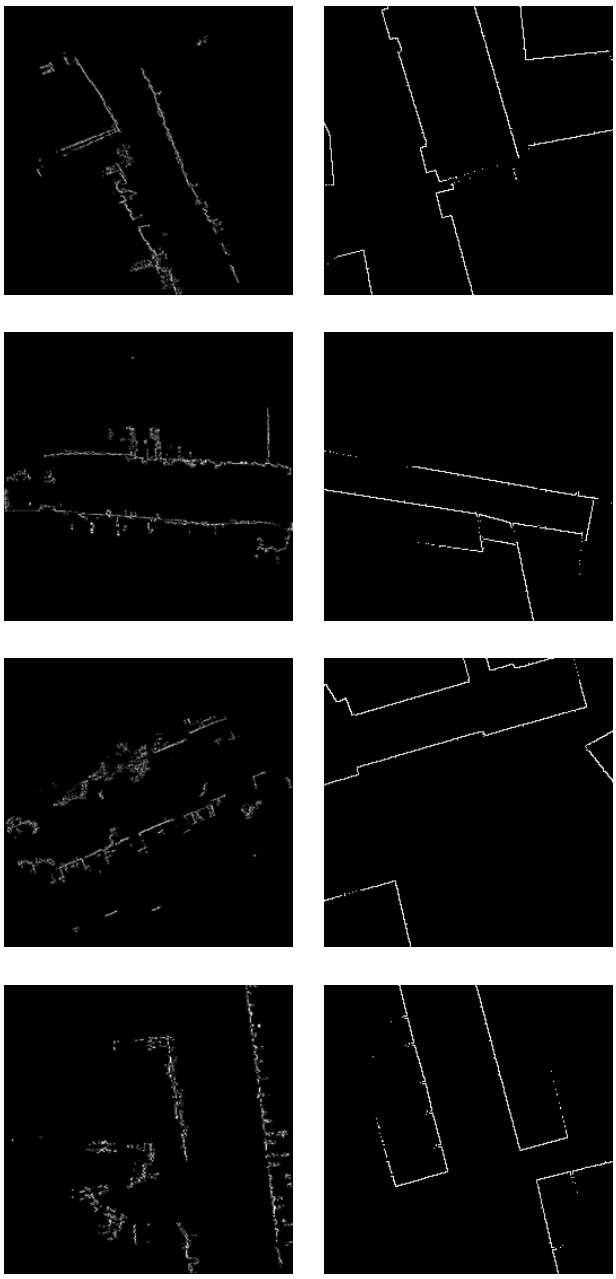


Figure 5.18: Left - query tile, Right - Predicted tile on raster. We note that the prediction manages to find structure quite similar to the kind of structure present in the query tiles. Note that as long as noise conforms somewhat to the true contours of the buildings around them, we get a good prediction that is close to the location of the query tile. Note that the scaling of the query tile and OSM tile doesn't exactly match up. This is mainly because we get our scale factor by scaling the metric distance travelled on road to the metric resolution of the typical OSM tile.

Examples with Localization errors between 1 and 2 Tile Widths

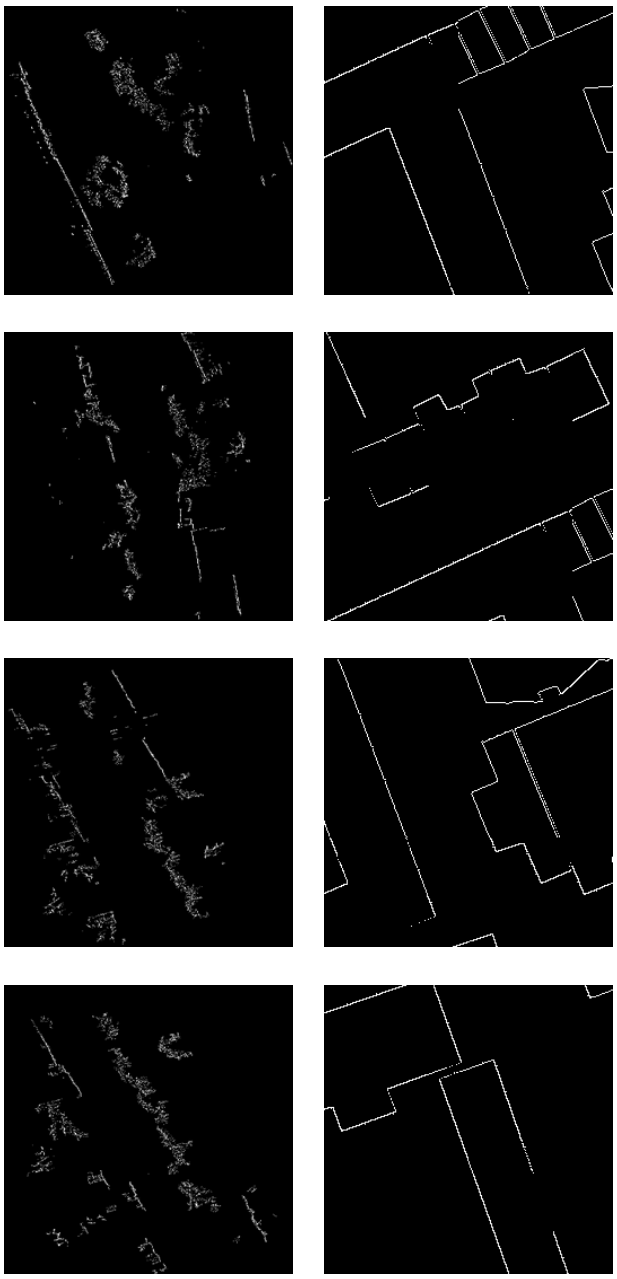


Figure 5.19: Left - query tile, Right - Predicted tile on raster. Looking at a few tiles with localization error of less than 2 tile widths, we observe that the network predicts tiles with similar structures which conform well to the original structure but are led astray by noise.

Examples with Localization errors between 2 and 3 Tile Widths

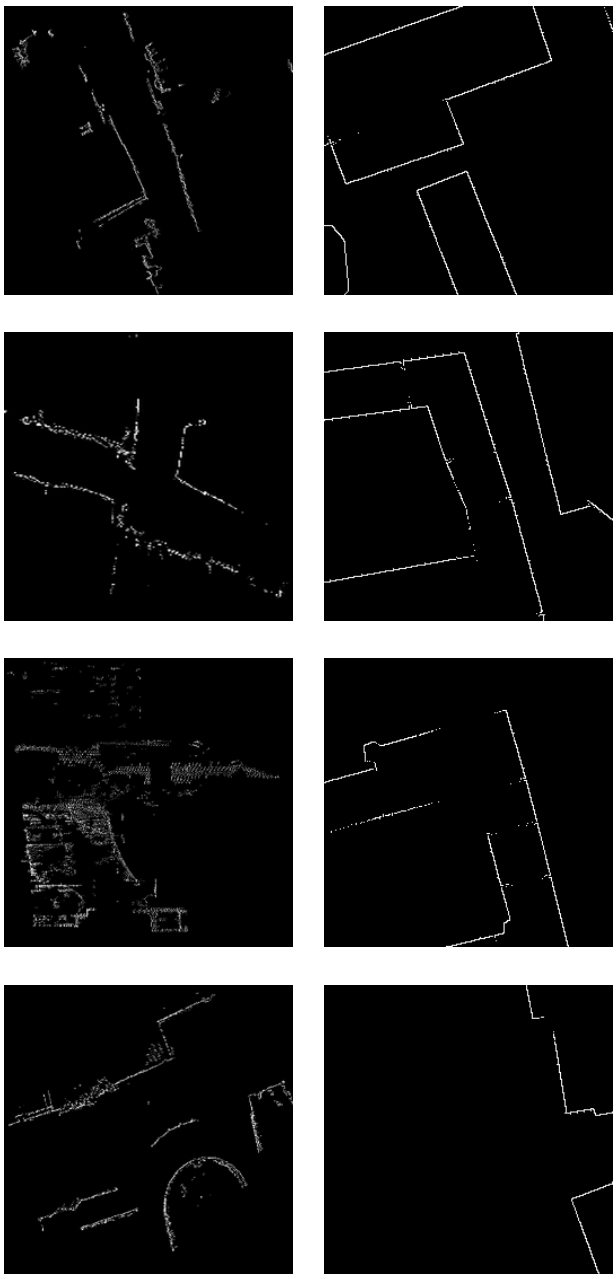


Figure 5.20: Left - query tile, Right - Predicted tile on raster. We still observe that the predictions are structurally similar to query tiles, but cannot be on point thanks to the noise contributed by trees segmented as buildings. Note that the scaling of the query tile and OSM tile doesn't exactly match up. This is mainly because we get our scale factor by scaling the metric distance travelled on road to the metric resolution of the typical OSM tile

Examples with Localization errors between 3 and 4 Tile Widths

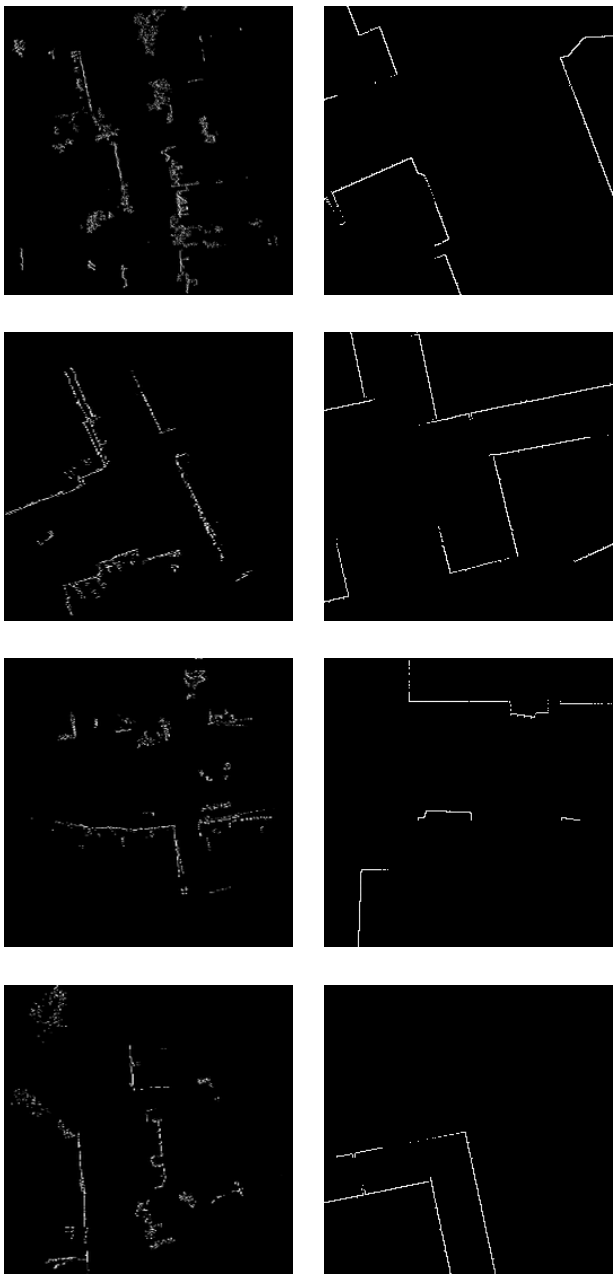


Figure 5.21: Left - query tile, Right - Predicted tile on raster. Structural similarity is still observable in the predictions, but noise confuses the network because it doesn't conform to the true building outline to give a good enough prediction.Note that the scaling of the query tile and OSM tile doesn't exactly match up. This is mainly because we get our scale factor by scaling the metric distance travelled on road to the metric resolution of the typical OSM tile

Examples with Localization errors between 4 and 5 Tile Widths

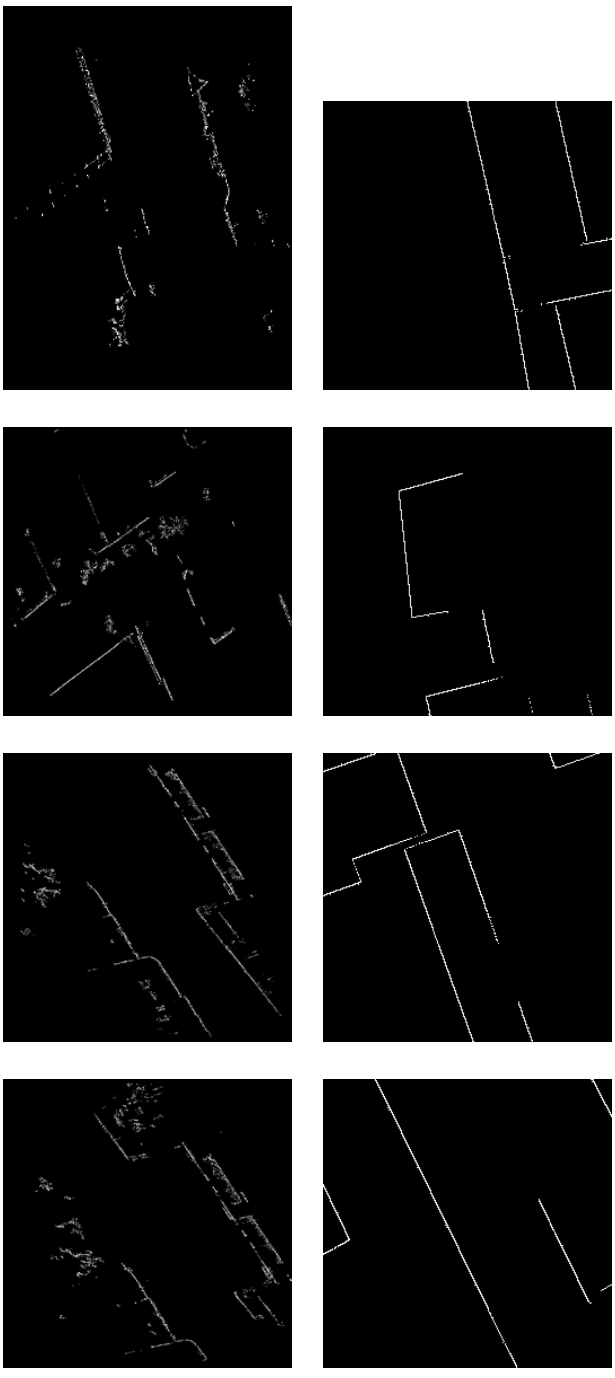


Figure 5.22: Left - query tile, Right - Predicted tile on raster. Structural similarity is still observable in the predictions, but noise confuses the network because it doesn't conform to the true building outline to give a good enough prediction. Note that the scaling of the query tile and OSM tile doesn't exactly match up. This is mainly because we get our scale factor by scaling the metric distance travelled on road to the metric resolution of the typical OSM tile

Examples with Localization errors between 5 and 6 Tile Widths

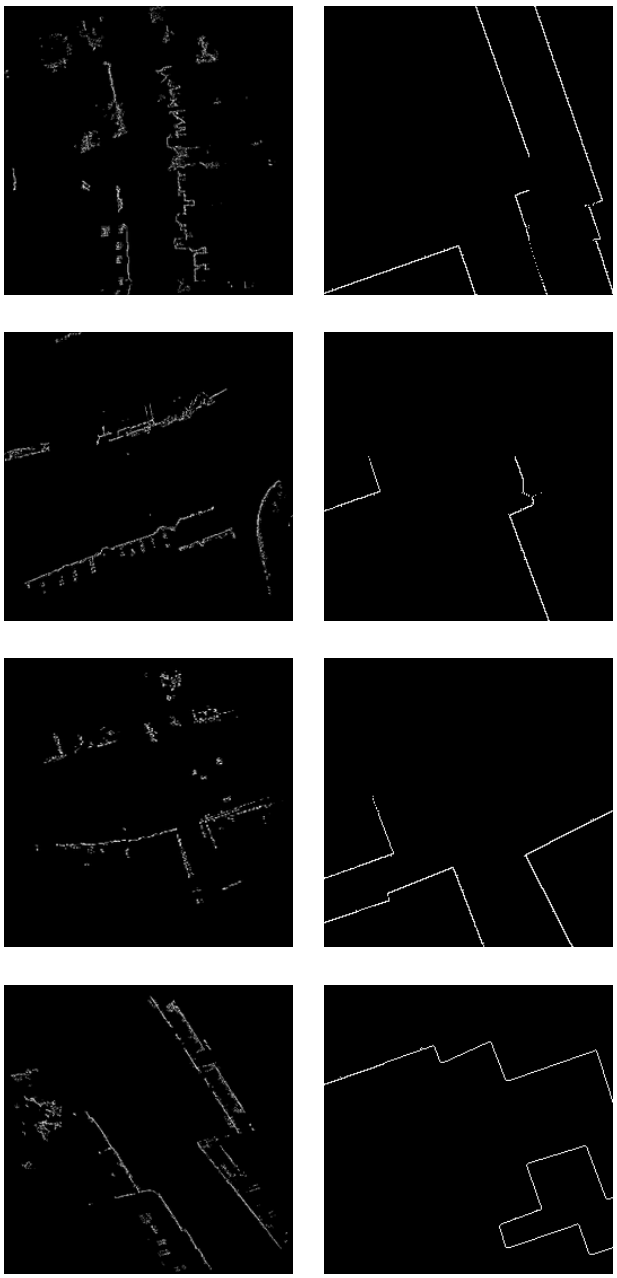


Figure 5.23: Left - query tile, Right - Predicted tile on raster. Structural similarity is still observable in the predictions, but noise confuses the network because it doesn't conform to the true building outline to give a good enough prediction. Note that the scaling of the query tile and OSM tile doesn't exactly match up. This is mainly because we get our scale factor by scaling the metric distance travelled on road to the metric resolution of the typical OSM tile

6 Conclusion

We have demonstrated in our work thus far the feasibility of our idea, which is localizing a car driving through an urban scenario directly on a map raster. We have shown that a pose regression network trained with overlapping tiles of some map area is able to localize the car's position, given a query map tile constructed from the scene. We've shown also a method to create a suitable training OSM tile set for a given area, taking into account issues such as the car's vield of view.

This being an initial formulation of the approach, we outlined how to construct the query map tile using labeled pointclouds. We've also described an approach to mitigating noise in the labeling of the pointcloud, which leads to an improved query map tile construction.

6.0.1 Future Work

The route we take to make the our query tiles exactly similar in content, scale and orientation to the Open Street Map tile at that location is not necessarily always robust. This is due to inaccuracies in transferring image labels to the pointcloud, and as mapping pixel/metre resolution of query tile to road resolution is not always accurate as we have access only to the distance travelled on the road.

As we have proved before in our evaluation section, such perturbations will lead to a noisy predictions.

In the future, we aim to improve the pipeline to function at a production level in the following possible ways:

- We can make the pose regression network robust to possibly noisy tiles. A possible way to deal with this problem is to train a Convolutional Neural Network to filter out the noisy top-down projection of the pointcloud to produce a clean query map tile, or train our Pose Regression network to be robust to noisy tiles during pose inference.

- To deal with inaccurate scaling, it is desirable to either incorporate data augmentation with scaling during training or rework the backbone network to be more robust to the scale variations of map tiles.

- Instead of focusing on being robust to noise, we can take the route of utilizing the now-popular pointcloud segmentation networks to directly get less noisier labelings of our LIDAR pointclouds. This will implicitly lead to more accurate query map-tile creations.

- Finally, we can do a more exhaustive hyperparameter search on the existing network parameters and tune the existing architecture of MapNet to better suit the specific problem of regressing pose off OSM tiles. The overlap between tiles while creating the training dataset will definitely have a strong influence on the results, and its effect must be explored more exhaustively.

Appendix

Bibliography

[Aln+20] Yara Ali Alnaggar, Mohamed Afifi, Karim Amer, and Mohamed Elhelw. *Multi Projection Fusion for Real-time Semantic Segmentation of 3D LiDAR Point Clouds*. 2020. arXiv: 2011.01974 [cs.CV] (cit. on p. 42).

[Ara+15] Relja Arandjelovic, Petr Gronát, Akihiko Torii, Tomás Pajdla, and Josef Sivic. "NetVLAD: CNN architecture for weakly supervised place recognition." In: *CoRR* abs/1511.07247 (2015). arXiv: 1511.07247. URL: http://arxiv.org/abs/1511.07247 (cit. on p. 20).

[Arm+17] Anil Armagan, Martin Hirzer, Peter M. Roth, and Vincent Lepetit. "Learning to Align Semantic Segmentation and 2.5D Maps for Geolocalization." In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. July 2017 (cit. on p. 30).

[AZ13] R. Arandjelovic and A. Zisserman. "All About VLAD." In: *2013 IEEE Conference on Computer Vision and Pattern Recognition*. 2013, pp. 1578–1585. DOI: 10.1109/CVPR.2013.207 (cit. on p. 19).

[BKC16] Vijay Badrinarayanan, Alex Kendall, and Roberto Cipolla. *SegNet: A Deep Convolutional Encoder-Decoder Architecture for Image Segmentation*. 2016. arXiv: 1511.00561 [cs.CV] (cit. on p. 16).

[Bra+17] Samarth Brahmbhatt, Jinwei Gu, Kihwan Kim, James Hays, and Jan Kautz. "MapNet: Geometry-Aware Learning of Maps for Camera Localization." In: vol. abs/1712.03342. 2017 (cit. on pp. 2, 33, 35).

[Che+17] Liang-Chieh Chen, George Papandreou, Iasonas Kokkinos, Kevin Murphy, and Alan L. Yuille. *DeepLab: Semantic Image Segmentation with Deep Convolutional Nets, Atrous Convolution, and Fully Connected CRFs*. 2017. arXiv: 1606.00915 [cs.CV] (cit. on p. 16).

[Das18] Sagarnil Das. "Simultaneous Localization and Mapping (SLAM) using RTAB-MAP." In: *CoRR* abs/1809.02989 (2018). arXiv: 1809.02989. URL: http://arxiv.org/abs/1809.02989 (cit. on p. 42).

[He+15] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. "Deep Residual Learning for Image Recognition." In: *CoRR* abs/1512.03385 (2015). arXiv: 1512.03385. URL: http://arxiv.org/abs/1512.03385 (cit. on p. 23).

[Hu+18] Sixing Hu, Mengdan Feng, Rang M. H. Nguyen, and Gim Hee Lee. "CVM-Net: Cross-View Matching Network for Image-Based Ground-to-Aerial Geo-Localization." In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. June 2018 (cit. on p. 30).

[KGC15] A. Kendall, M. Grimes, and R. Cipolla. "PoseNet: A Convolutional Network for Real-Time 6-DOF Camera Relocalization." In: *2015 IEEE International Conference on Computer Vision (ICCV)*. Dec. 2015, pp. 2938–2946. DOI: 10.1109/ICCV.2015.336 (cit. on pp. 2, 22, 35).

[KSH12] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. "ImageNet Classification with Deep Convolutional Neural Networks." In: *Advances in Neural Information Processing Systems*. Ed. by F. Pereira, C. J. C. Burges, L. Bottou, and K. Q. Weinberger. Vol. 25. Curran Associates, Inc., 2012, pp. 1097–1105. URL: https://proceedings.neurips.cc/paper/2012/file/c399862d3b9d6b76c8436e924a68c45b-Paper.pdf (cit. on p. 14).

[Lam+20] John Lambert, Zhuang Liu, Ozan Sener, James Hays, and Vladlen Koltun. "MSeg: a composite dataset for multi-domain semantic segmentation." In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 2020, pp. 2879–2888 (cit. on p. 42).

[LBH13] T. Lin, S. Belongie, and J. Hays. "Cross-View Image Geolocalization." In: *2013 IEEE Conference on Computer Vision and Pattern Recognition*. 2013, pp. 891–898. DOI: 10.1109/CVPR.2013.120 (cit. on p. 27).

[LRB15] Wei Liu, Andrew Rabinovich, and Alexander C. Berg. *ParseNet: Looking Wider to See Better*. 2015. arXiv: 1506.04579 [cs.CV] (cit. on p. 16).

[LSD15] Jonathan Long, Evan Shelhamer, and Trevor Darrell. *Fully Convolutional Networks for Semantic Segmentation*. 2015. arXiv: 1411.4038 [cs.CV] (cit. on p. 15).

[Mad+17] Will Maddern, Geoff Pascoe, Chris Linegar, and Paul Newman. "1 Year, 1000km: The Oxford RobotCar Dataset." In: *The International Journal of Robotics Research (IJRR)* 36.1 (2017), pp. 3–15. DOI: 10.1177/0278364916679498. eprint: http://ijr.sagepub.com/content/early/2016/11/28/0278364916679498.full.pdf+html. URL: http://dx.doi.org/10.1177/0278364916679498 (cit. on pp. 34, 38, 39, 57).

[May+16] Nikolaus Mayer, Eddy Ilg, Philip Hausser, Philipp Fischer, Daniel Cremers, Alexey Dosovitskiy, and Thomas Brox. "A Large Dataset to Train Convolutional Networks for Disparity, Optical Flow, and Scene Flow Estimation." In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. June 2016 (cit. on p. 26).

[MMT15] Raul Mur-Artal, J. M. M. Montiel, and Juan D. Tardós. "ORB-SLAM: a Versatile and Accurate Monocular SLAM System." In: *CoRR* abs/1502.00956 (2015). URL: http://dblp.uni-trier.de/db/journals/corr/corr1502.html#Mur-ArtalMT15 (cit. on p. 42).

[NS06] D. Nister and H. Stewenius. "Scalable Recognition with a Vocabulary Tree." In: *2006 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'06)*. Vol. 2. 2006, pp. 2161–2168. DOI: 10.1109/CVPR.2006.264 (cit. on p. 19).

[Ope17] OpenStreetMap contributors. *Planet dump retrieved from https://planet.osm.org*. https://www.openstreetmap.org. 2017 (cit. on pp. 34, 36, 37, 39, 40, 47, 50).

[Qi+16] Charles R. Qi, Hao Su, Kaichun Mo, and Leonidas J. Guibas. *PointNet: Deep Learning on Point Sets for 3D Classification and Segmentation*. cite arxiv:1612.00593. 2016. URL: http://arxiv.org/abs/1612.00593 (cit. on p. 42).

[Sat+19] T. Sattler, Q. Zhou, M. Pollefeys, and L. Leal-Taixé. "Understanding the Limitations of CNN-Based Absolute Camera Pose Regression." In: *2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. 2019, pp. 3297–3307 (cit. on p. 2).

[Shi+19] Yujiao Shi, Xin Yu, Liu Liu, Tong Zhang, and Hongdong Li. "Optimal Feature Transport for Cross-View Image Geo-Localization." In: *CoRR* abs/1907.05021 (2019). arXiv: 1907.05021. URL: http://arxiv.org/abs/1907.05021 (cit. on p. 28).

[Sho+13] Jamie Shotton, Ben Glocker, Christopher Zach, Shahram Izadi, Antonio Criminisi, and Andrew Fitzgibbon. "Scene Coordinate Regression Forests for Camera Relocalization in RGB-D Images." In: *Proc. Computer Vision and Pattern Recognition (CVPR)*. IEEE, June 2013 (cit. on p. 22).

[SLK12] Torsten Sattler, Bastian Leibe, and Leif Kobbelt. "Towards Fast Image-Based Localization on a City-Scale." In: *Outdoor and Large-Scale Real-World Scene Analysis*. Ed. by Frank Dellaert, Jan-Michael Frahm, Marc Pollefeys, Laura Leal-Taixé, and Bodo Rosenhahn. Berlin, Heidelberg: Springer Berlin Heidelberg, 2012 (cit. on p. 18).

[SSP19] T. Schöps, T. Sattler, and M. Pollefeys. "BAD SLAM: Bundle Adjusted Direct RGB-D SLAM." In: *2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. 2019, pp. 134–144. DOI: 10.1109/CVPR.2019.00022 (cit. on p. 42).

[SZ15] K. Simonyan and Andrew Zisserman. "Very Deep Convolutional Networks for Large-Scale Image Recognition." In: *CoRR* abs/1409.1556 (2015) (cit. on pp. 14, 29).

[Sze+15] Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. "Going Deeper with Convolutions." In: *Computer Vision and Pattern Recognition (CVPR)*. 2015. URL: http://arxiv.org/abs/1409.4842 (cit. on p. 14).

[Val+17] A. Valada, J. Vertens, A. Dhall, and W. Burgard. "AdapNet: Adaptive semantic segmentation in adverse environmental conditions." In: *2017 IEEE International Conference on Robotics and Automation (ICRA)*. 2017, pp. 4644–4651. DOI: 10.1109/ICRA.2017.7989540 (cit. on p. 16).

[VRB18] Abhinav Valada, Noha Radwan, and Wolfram Burgard. "Deep Auxiliary Learning for Visual Localization and Odometry." In: *CoRR* abs/1803.03642 (2018). arXiv: 1803.03642. URL: http://arxiv.org/abs/1803.03642 (cit. on p. 23).

[WJ15] S. Workman and N. Jacobs. "On the location dependence of convolutional neural network features." In: *2015 IEEE Conference on Computer Vision and Pattern Recognition Workshops (CVPRW)*. 2015, pp. 70–78. DOI: 10.1109/CVPRW.2015.7301385 (cit. on p. 27).

[YK16] Fisher Yu and Vladlen Koltun. *Multi-Scale Context Aggregation by Dilated Convolutions*. 2016. arXiv: 1511.07122 [cs.CV] (cit. on p. 16).

[Zha+17] Hengshuang Zhao, Jianping Shi, Xiaojuan Qi, Xiaogang Wang, and Jiaya Jia. *Pyramid Scene Parsing Network*. 2017. arXiv: 1612.01105 [cs.CV] (cit. on p. 16).

[ZPK18] Qian-Yi Zhou, Jaesik Park, and Vladlen Koltun. "Open3D: A Modern Library for 3D Data Processing." In: *arXiv:1801.09847* (2018) (cit. on p. 43).