Low-cost Real-Time Learning-based Localization for Autonomous Systems

Rahul Mangharam https://orcid.org/0000-0003-2539-896X
University of Pennsylvania
200 S 33rd St, Philadelphia PA 19104

FINAL REPORT

October 29, 2024

DISCLAIMER

The contents of this report reflect the views of the authors, who are responsible for the facts and the accuracy of the information presented herein. This document is disseminated in the interest of information exchange. The report is funded, partially or entirely, under [grant number 69A3552344811] from the U.S. Department of Transportation's University Transportation Centers Program. The U.S. Government assumes no liability for the contents or use thereof.

1. Report No.	2. Government Accession No.	3. Recipient's Catalog No.
441		

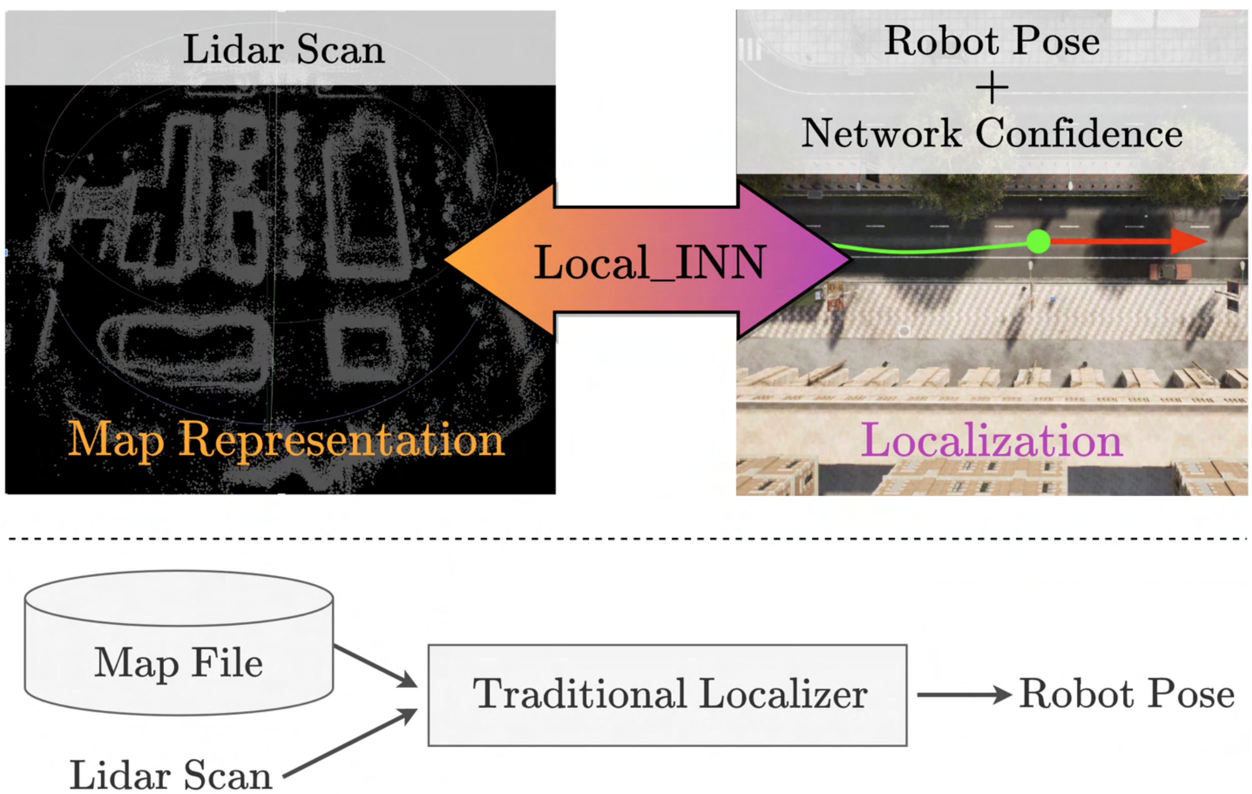
4. Title and Subtitle Low-cost Real-Time Learning-based Localization for Autonomous Systems	**5. Report Date** October 29,2024 **6. Performing Organization Code**
7. Author(s) Rahul Mangharam: https://orcid.org/0000-0003-2539-896X	**8. Performing Organization Report No.** **441**
9. Performing Organization Name and Address University of Pennsylvania 200 S 33rd St, Philadelphia PA 19104	**10. Work Unit No.** **11. Contract or Grant No.** Federal Grant No. 69A3552344811
12. Sponsoring Agency Name and Address Safety21 University Transportation Center Carnegie Mellon University 5000 Forbes Avenue Pittsburgh, PA 15213	**13. Type of Report and Period Covered** Final Report (July 1, 2023-June 30, 2024) **14. Sponsoring Agency Code** USDOT
15. Supplementary Notes Conducted in cooperation with the U.S. Department of Transportation, Federal Highway Administration.	

16. Abstract			
Robot localization is the problem of finding a robot's pose using a map and sensor measurements, like LiDAR scans or camera images. It is crucial for any moving autonomous vehicle to interact with the physical world correctly. However, finding injective mappings between measurements and poses is difficult because sensor measurements from multiple distant poses can be similar. To solve this ambiguity, Monte Carlo Localization (MCL), the widely adopted method, uses random hypothesis sampling and sensor measurement updates to infer the pose. Other common approaches are to use Bayesian filtering or to find better-distinguishable global descriptors on the map. Recent developments in localization research usually propose better measurement models or feature extractors within these frameworks. On contrary, this project we propose a radically new approach to frame the localization problem as an ambiguous inverse problem and solve it with an invertible neural network (INN). We claim that INN is naturally suitable for the localization problem with many benefits, in terms of high accuracy (within 0.25m for city-scale maps), high-speed operation (>150Hz) and operates on low-cost embedded system hardware. We will demonstrate this on point-cloud and camera datasets with evaluation on indoor and outdoor localization benchmarks, and also deploy it on 1/10th scale and 1/2 scale autonomous vehicles to show real-time and scalable operation.			
17. Key Words Invertible Neural Network, Autonomous vehicles, robotics, computational thinking, machine learning, control, simulation		**18. Distribution Statement** No restrictions. This document is available through the National Technical Information Service, Springfield, VA 22161. Enter any other agency mandated distribution statements. Remove NTIS statement if it does not apply.	
19. Security Classif. (of this report) Unclassified	**20. Security Classif. (of this page)** Unclassified	**21. No. of Pages** 24	**22. Price** Refers to the price of the report. Leave blank unless applicable.

Localization with Invertible Neural Networks (LocalINN and PoseINN)

1. Introduction

Localization is a critical problem for autonomous systems, whether for mobile robotics, augmented reality, or self-driving vehicles. The challenge lies in accurately determining the vehicle or robot's position within an environment, using sensor data such as LiDAR or cameras. Traditional methods, such as Monte Carlo Localization (MCL) and Bayesian filters, face limitations in computational cost and speed, especially when applied to real-time systems.

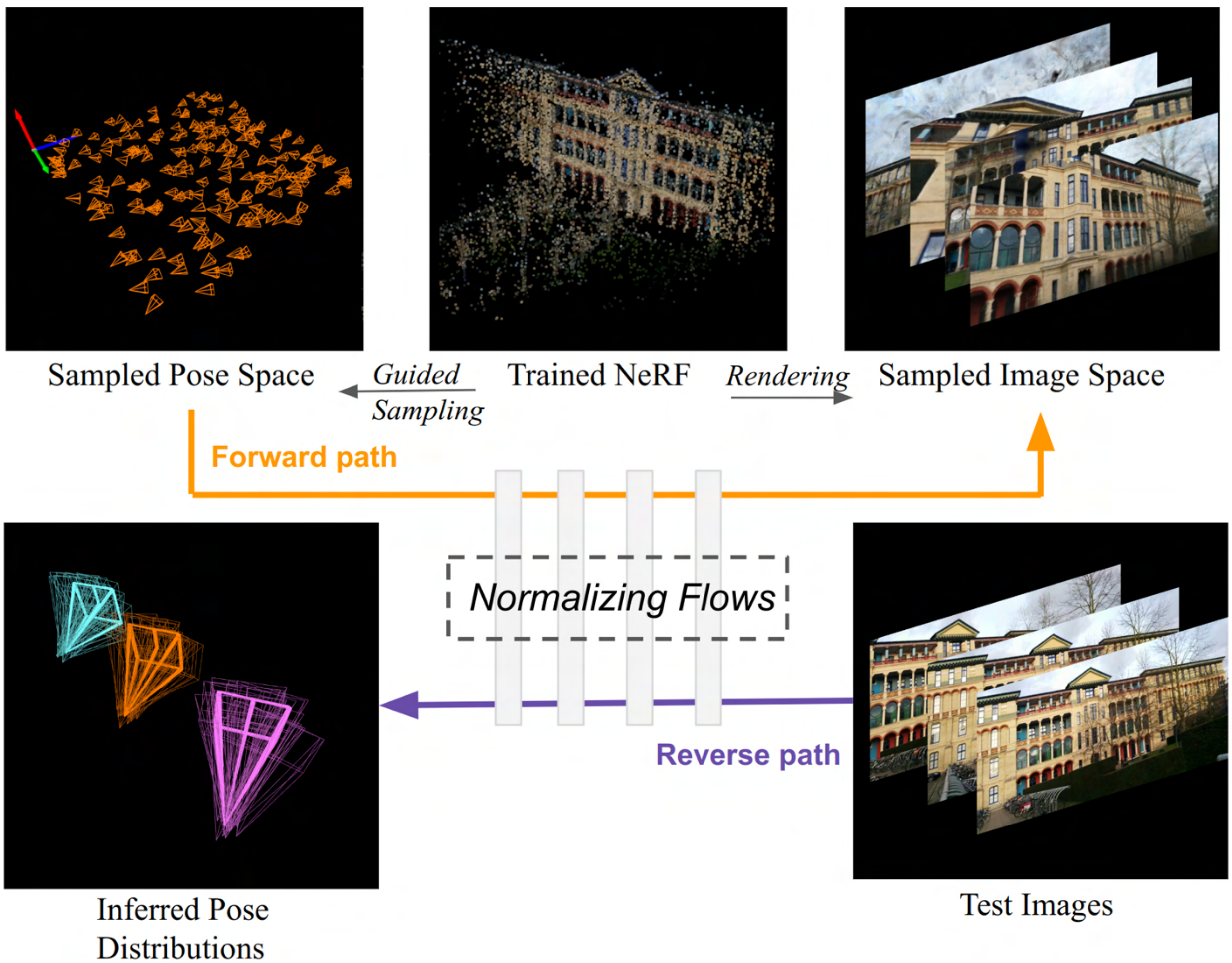


In this report, we describe two novel methods—LocalINN and PoseINN—that leverage Invertible Neural Networks (INNs) to improve localization accuracy and speed. These approaches compress map representations and utilize latent space sampling for uncertainty estimation. LocalINN focuses on LiDAR-based localization, while PoseINN extends the framework to visual-based localization using camera data. The report summarizes the problem, approach, methodology, findings, conclusions, and recommendations based on extensive research and experimentation.

2. Problem Statement

Current robot localization methods often require high computational power, large amounts of training data, and expensive sensors. Localization with LiDAR or camera systems has inherent ambiguities due to similarities in measurements from different positions, making it difficult to map sensor readings directly to unique poses.

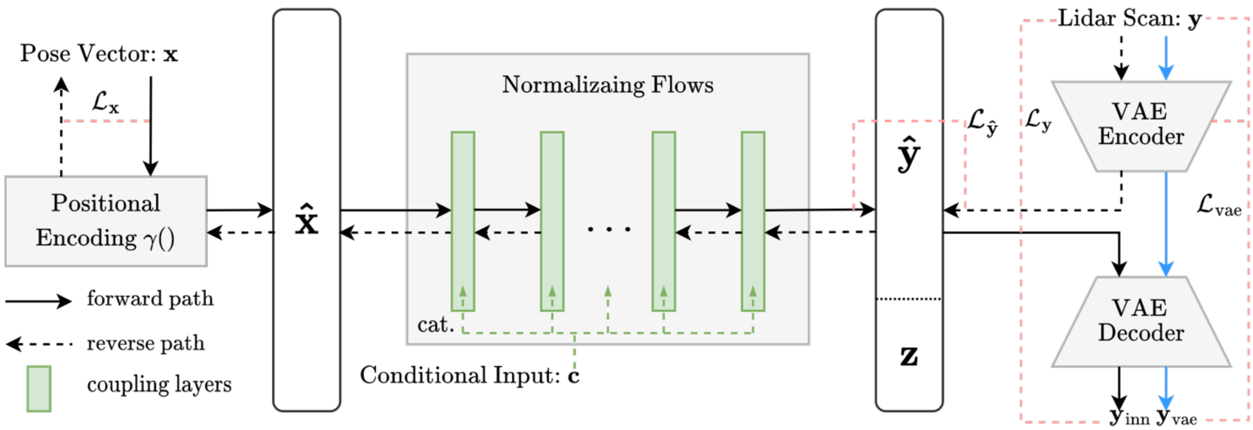
LocalINN and PoseINN address these issues by framing localization as an ambiguous inverse problem, solving it with INNs. These approaches provide high accuracy (within 0.25 meters for city-scale maps), fast processing speeds (up to 270Hz), and operate on low-cost hardware, making them suitable for real-time autonomous applications.



3. Approach

Both LocalINN and PoseINN use Invertible Neural Networks (INNs) to transform sensor data (LiDAR or camera images) into pose estimations. These neural networks learn a bijective mapping between the source (sensor data) and target distributions (robot poses), with an emphasis on fast runtime and uncertainty estimation. The key difference between the two approaches lies in the type of sensor data used—LiDAR for LocalINN and visual camera data for PoseINN.

1. **LocalINN**: Compresses a 2D/3D map into the neural network, allowing the system to localize without needing to access a separate map file. LocalINN outputs a full posterior distribution of poses with covariance, which enhances robustness in challenging environments.
2. **PoseINN**: Uses NeRF (Neural Radiance Fields) to generate synthetic views from cameras and maps the camera images to corresponding poses. PoseINN provides a fast data preparation pipeline and achieves real-time visual localization on embedded platforms like mobile robots.



4. Methodology

The development and testing of LocalINN and PoseINN followed these key steps:

1. **Map Compression and Data Preparation**: LocalINN learns the implicit representation of a 2D/3D map and compresses it into the neural network. PoseINN uses NeRF to generate synthetic camera images, which are used to train the model for visual localization.
2. **Training and Testing**: The models were trained on various benchmark datasets. LocalINN was tested with LiDAR data, while PoseINN was tested with both simulated and real-world camera data.
3. **Uncertainty Estimation**: Both models provide pose estimations along with uncertainty values, which are integrated into an Extended Kalman Filter (EKF) for more robust results.
4. **Real-Time Deployment**: Both LocalINN and PoseINN were deployed on mobile robots, including the F1TENTH autonomous vehicle platform, demonstrating their capability for real-time operation in complex environments.

5. Findings

1. **LocalINN Performance**: In tests using 2D and 3D LiDAR data, LocalINN achieved comparable or superior accuracy to particle filter methods, with much lower latency (up to 270Hz using TensorRT). LocalINN showed robust performance even in challenging environments, with fast global localization recovery.

2. **PoseINN Performance**: PoseINN, when tested on mobile robots, provided real-time camera-based localization with lower computational overhead than traditional methods. The model's performance was comparable to state-of-the-art systems but required much less training data and processing power.
3. **Uncertainty Estimation**: Both models output pose distributions, which provide more reliable results when fused with other sensor data. The models' ability to output confidence values enables more accurate navigation in uncertain or dynamic environments.

6. Conclusions

LocalINN and PoseINN represent a significant advancement in real-time localization for autonomous systems. By leveraging INNs, these models reduce the need for large computational resources, enable fast and accurate localization, and provide uncertainty estimation for safer autonomous operations. The integration of NeRF in PoseINN further demonstrates how synthetic data generation can be used to reduce training time while maintaining accuracy.

7. Recommendations

1. **Further Development**: Future work should focus on expanding these methods to more challenging environments and improving their robustness in dynamic environments with moving objects.
2. **Wider Deployment**: Encourage wider adoption of these methods in autonomous vehicle research, including applications in self-driving cars, drones, and augmented reality.
3. **Community Collaboration**: Developing an open-source toolkit for LocalINN and PoseINN would enable broader community collaboration and innovation.

8. Project Outputs and Documentation

1. **Final Report URL(s) or PDFs**: https://ieeexplore.ieee.org/document/10161015
2. **Dataset URL(s) and Descriptive Metadata**: https://github.com/zzangupenn/Local_INN
3. **ORCIDs for Project Investigators**:
    1. Zirui Zang: ORCID link
    2. Rahul Mangharam: ORCID link

Local_INN: Implicit Map Representation and Localization with Invertible Neural Networks

Zirui Zang, Hongrui Zheng, Johannes Betz, Rahul Mangharam

***Abstract*—Robot localization is an inverse problem of finding a robot's pose using a map and sensor measurements. In recent years, Invertible Neural Networks (INNs) have successfully solved ambiguous inverse problems in various fields. This paper proposes a framework that approaches the localization problem with INN. We design a network that provides implicit map representation in the forward path and localization in the inverse path. By sampling the latent space in evaluation, Local_INN outputs robot poses with covariance, which can be used to estimate the uncertainty. We show that the localization performance of Local_INN is on par with current methods with much lower latency. We show detailed 2D and 3D map reconstruction from Local_INN using poses exterior to the training set. We also provide a global localization algorithm using Local_INN to tackle the kidnapping problem.**

I. Introduction

Robot localization is the problem of finding a robot's pose using a map and sensor measurements, like LiDAR scans. It is crucial for any moving robot to interact with the physical world correctly. However, finding injective mappings between measurements and poses is difficult because sensor measurements from multiple distant poses can be similar.

To solve this ambiguity, Monte Carlo Localization (MCL) [1], [2], the widely adopted method, uses random hypothesis sampling and sensor measurement updates to infer the pose. Other common approaches are to use Bayesian filtering [3] or to find better-distinguishable global descriptors on the map [4], [5]. Recent developments in localization research usually propose better measurement models or feature extractors within these frameworks. On contrary, this paper proposes a new approach to frame the localization problem as an ambiguous inverse problem and solve it with an invertible neural network (INN). We claim that INN is naturally suitable for the localization problem with many benefits, as we will show in this paper.

Robot localization is an inverse problem, which is when we are given a set of observations and try to find the causal factors. In a well-modeled environment, it's easier to calculate the expected observations if given the causal factors. In the context of LiDAR-based localization, the robot's pose in the environment causes the particular scan measurements. In addition, when given a map, we can easily simulate LiDAR scans from any pose on the map.

Invertible neural networks such as normalizing flows [6]–[9] have been used to solve inverse problems in various fields [10]–[15]. It learns a bijective mapping between the source and target distributions with a series of invertible transformations. It uses a latent space to capture the lost ambiguous information during training. We use pose-scan data pairs to train such a bijective mapping. The forward path is from pose to scan and the reverse path is from scan to pose. Because INNs require the same input and output dimensions, we use a Variational Autoencoder (VAE) [16] to reduce the dimension of the lidar scans and use Positional Encoding [17] to augment the dimension of the poses. With the help of conditional inputs, we can reduce the ambiguity of the inverse problem. In our case, we use zones in the map calculated from the previous pose of the robot as conditional input into the INN. During the evaluation, we sample the latent space to find the full posterior distribution of the pose, given a sensor measurement. We validated our method in localization experiments with 2D and 3D LiDARs, both in simulation and with real data. To summarize, this paper has four major contributions:

1) **Map Compression**: Local_INN provides an implicit map representation and a localization method within one neural network. Map files are no longer needed when localizing.
2) **Uncertainty Estimation**: Local_INN outputs not just a pose but a distribution of inferred poses, the covariance of which can be used as the confidence of the neural network when fusing with other sensors, enhancing the overall robustness.
3) **Fast and Accurate**: We demonstrate that the localization performance of Local_INN is comparable to particle filter at slow speed and better at high speed with much lower

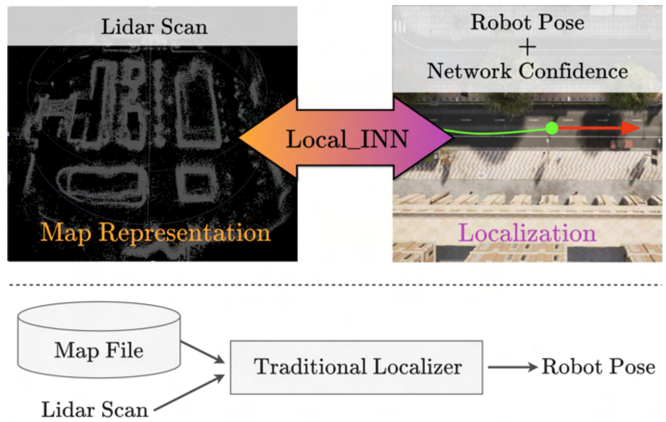


Fig. 1. Local_INN is a framework of localization with invertible neural networks. Compared to current localization methods, Local_INN stores map information within the neural network. Evaluation of Local_INN in the forward direction gives compressed map information, and in the reverse direction gives accurate localization with fast runtime and uncertainty estimation.

This work was supported in part by NSF CCRI tel:1925587 and DARPA FA8750-20-C-0542 (Systemic Generative Engineering). The views, opinions, and/or findings expressed are those of the author(s) and should not be interpreted as representing the official views or policies of the Department of Defense or the U.S. Government.

All authors are with the University of Pennsylvania, Department of Electrical and Systems Engineering, 19104, Philadelphia, PA, USA. Emails: {zzang, hongruiz, joebetz, rahulm}@seas.upenn.edu

latency with 2D LiDAR experiments.

4) **Ability to Generalize**: We demonstrate that the framework of Local_INN can learn complex 3D open-world environments and provides accurate localization. We also provide an algorithm for global localization with Local_INN.

II. Related Work

Local_INN sits at the intersection of two research fields: Localization and Normalizing Flows. In this section, we will briefly introduce both fields.

A. Lidar-based Localization

Monte Carlo Localization (MCL) [1], ever since its introduction, has been a popular localization framework for its reliable performance and the modularity to swap the motion or measurement model with any desired method. Many developments in localization seek to improve within the framework of MCL. [18]–[20] Outside the framework of MCL, people have used methods such as Bayesian inference [21], RNNs [22], [23], global descriptors [24], [25], or combining them [26]. We propose Local_INN as a new framework of solve the problem.

For learning-based localization methods, uncertainty estimations of the neural networks become a challenge. There are efforts to approximate the uncertainty [27]–[29], but it hasn't been widely applied. Local_INN comes naturally with an uncertainty estimation due to the use of normalizing flows.

Large map size is also becoming a burden as pointed out by [30]. After the advent of NeRF [31], it was clear that neural networks are very capable of implicitly representing spatial information. There are developments in using neural networks for implicit map representation in the SLAM pipeline [32], [33]. Local_INN builds on that while providing a method of localization.

B. Normalizing Flows

A normalizing flow is a series of invertible transformations that gradually transform a source data distribution into a target data distribution. Methods of achieving such bijective mappings have been developing rapidly in recent years [9]. Real-valued non-volume preserving (RealNVP) transformations introduced by Dinh et al. [7] use coupling layers that are efficient to compute in both forward and reverse processes. Although newer normalizing flows have better expressiveness [34]–[36], we choose to use RealNVP for its efficiency.

The framework of solving ambiguous inverse problems using normalizing flows was introduced by Ardizonne et al. [10] and was later extended by [11], [37] to include a conditional input that is concatenated to the vectors inside the coupling layers. They proposed to use a latent variable to encode the lost information in training due to the ambiguity of the problem. During the evaluation, repeatedly sampling the latent variable can give the full posterior distribution given the input. In this paper, we added a VAE to the framework so that we can use high-dimensional input. The use of latent variables gives us distributions of estimated poses, which we can use to calculate the covariance.

III. Methodology

LiDARs are widely used in moving robots and autonomous vehicles. 2D or 3D LiDARs produce one or multiple arrays of range distances with each value in the array being the distance from the robot to the closest obstacle at a certain angle. The localization problem with LiDAR is: given a LiDAR scan, find the robot's $[x, y]$ coordinates on the map and its heading θ relative to the x axis of the map.

We use normalizing flow to find a bijective mapping between a robot's pose vector $\mathbf{x} \in \mathbb{R}^3$ on the map and LiDAR scan vector $\mathbf{y} \in \mathbb{R}^{\text{angle}}$ with a latent vector $\mathbf{z} \in \mathbb{R}^6$. The forward path of the localization problem is easy, so we can simulate an infinite amount of pose-scan data pairs for training by randomly sampling the state space. We use a rounded pose (as in equation 2) of the robot to produce the conditional input $\mathbf{c} \in \mathbb{R}^3$ in the INN. This rounded pose can be computed during testing by rounding the robot's previous pose. Because INN requires the same input and output dimension, we use positional encoding to augment the pose vector $\mathbf{x}$ to $\hat{\mathbf{x}} \in \mathbb{R}^{6L}$, where L is the level of the sine-cosine positional encoding [17]. On the LiDAR scan side, we use a VAE [16] to encode the LiDAR scan $\mathbf{y}$ to $\hat{\mathbf{y}} \in \mathbb{R}^{6L-6}$, which is concatenated with latent vector $\mathbf{z} \sim \mathcal{N}(0, 1)$. We use the latent vector to catch the full posterior distribution of $\mathbf{x}$ conditioned on $\mathbf{c}$ given $\mathbf{y}$. This can later be used to sample the covariance of the inferred pose vector.

A. Conditional Normalizing Flow

Normalizing flows contain a series of invertible transformations. We use the affine coupling block architecture introduced in Real-NVP [7] and extended by [11], [37] to incorporate a conditional input. The forward path of a single coupling block is:

$$\begin{aligned} \mathbf{v}_1 &= \mathbf{u}_1 \odot \exp(s_2(\mathbf{u}_2, \hat{\mathbf{c}})) + t_2(\mathbf{u}_2, \hat{\mathbf{c}}), \\ \mathbf{v}_2 &= \mathbf{u}_2 \odot \exp(s_1(\mathbf{v}_1, \hat{\mathbf{c}})) + t_1(\mathbf{v}_1, \hat{\mathbf{c}}). \end{aligned} \tag{1}$$

The input $\mathbf{u}$ is split into two halves $\mathbf{u}_1$ and $\mathbf{u}_2$, which undergo affine transformations with scale coefficient s_i and translation coefficient t_i for $i \in \{1, 2\}$. Here $\odot$ is element-wise multiplication. The outputs $\mathbf{v}_1$ and $\mathbf{v}_2$ are then concatenated together before exiting this coupling block. The exponential function here is to eliminate zero outputs, which ensures invertibility. In the reverse direction, given $\mathbf{v}_1$ and $\mathbf{v}_2$, this structure is easily invertible without any computational overheads. Therefore, s_i and t_i are not required to be invertible and can be learned with neural networks. Multiple coupling blocks are connected to increase the expressiveness of the normalizing flows. After each coupling block, there is a predefined random permutation to shuffle the variables so that the splitting of the input vector is different for each block. We followed [15] to use two layers of MLP with ReLU activation in each affine coupling block and used a parameterized soft clamping mechanism to prevent instabilities. Let's denote the forward and reverse path of the INN network with $h_{\text{inn}}^{forward}$, $h_{\text{inn}}^{reverse}$.

To deal with the inverse ambiguity due to map symmetry, a rounded pose computed from the robot's previous pose $\mathbf{x}^{\text{pre}}$

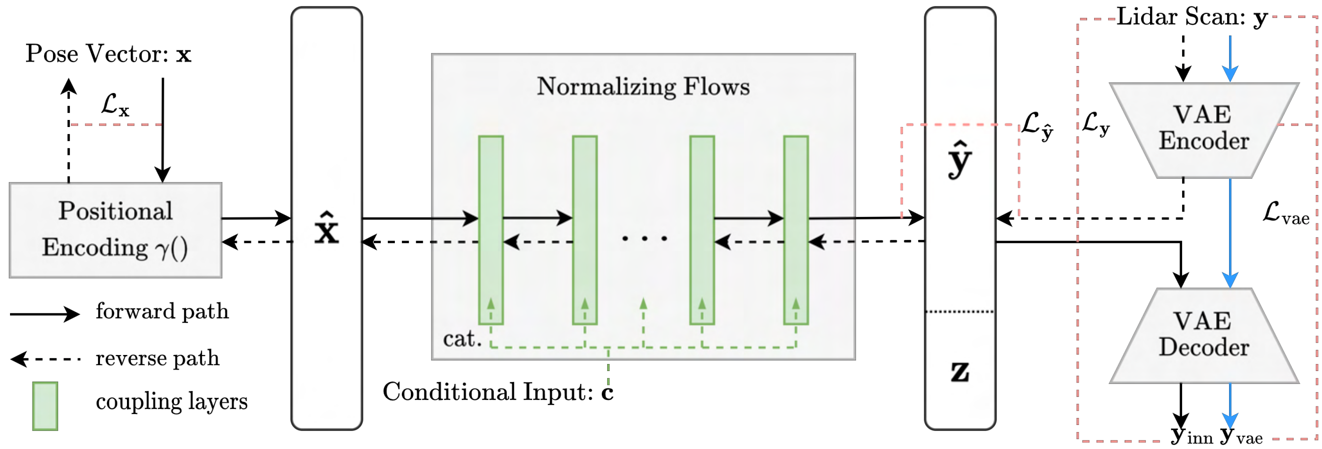


Fig. 2. Network Structure of the Local_INN. The forward path (solid arrows) is from pose to LiDAR scan. The reverse path (dashed arrows) is from LiDAR scan to robot pose. Conditional input is calculated from the robot's previous pose. The INN used in this paper has 6 coupling layers and the VAE encoder and decoder have 2 layers of MLPs for 2D LiDARs and plus 6 layers of 2D convolutions for 3D LiDARs.

is passed through a positional encoding $\gamma(\cdot)$, then encoded by a separate MLP h_{cond} before concatenating to $\mathbf{u}_i$ or $\mathbf{v}_i$ in the coupling block:

$$\mathbf{c} = \frac{\lceil N\mathbf{x}^{\text{pre}} \rfloor}{N}, \hat{\mathbf{c}} = h_{\text{cond}}(\gamma(\mathbf{c})). \tag{2}$$

During training, $\mathbf{x}^{\text{pre}}$ is approximated by adding a zero mean Gaussian noise to the ground truth pose:

$$\mathbf{x}^{\text{pre}}_{\text{training}} = \mathbf{x} + \delta, \delta \sim \mathcal{N}(0, \sigma^2). \tag{3}$$

The rounded previous states essentially divide the state space into N^3 zones, and which zone the robot previously existed in is provided to the INN as conditional input. The Gaussian noise during training ensures that it's okay for the $\mathbf{x}^{\text{pre}}$ near zone boundaries to be rounded into either neighboring zones. Depending on the map, σ^2 for $[x, y, \theta]$ and integer parameter N needs to be picked. We picked σ^2 around 0.5 meters and $N = 10$ for all our experiments, which means the zones are quite large for the 3D maps.

B. Positional Encoding

Positional encoding was used in [17] [31] to boost the performance of the neural network in fitting high-frequency information. The positional encoding $\gamma(\cdot)$ we used maps from $\mathbb{R}$ to $\mathbb{R}^{2L}$ with increasing frequencies:

$$\begin{aligned}\gamma(p) = (\,&\sin(2^0\pi p), \cos(2^0\pi p), \ldots, \\ &\sin(2^{L-1}\pi p), \cos(2^{L-1}\pi p)).\end{aligned} \tag{4}$$

When applied to pose vectors, function $\gamma(\cdot)$ is applied separately to $[x, y, \theta]$. Pose vector $\mathbf{x}$ is encoded with $L = 10$ and the conditional input is encoded with $L = 1$. All variables are normalized to $[0, 1)$ before being applied to $\gamma(\cdot)$. We observed that adding positional encoding directly helps the forward path by augmenting the 3-dimensional input, which in turn helps the reverse training as well.

C. Variational Autoencoder

LiDARs produce hundreds to thousands of range data points per channel. Due to the input and output dimension requirement of the INN, putting everything into the INN would vastly increase the size of the network without proportional benefit. On the other hand, sub-sampling LiDAR scans increase susceptibility to noisy or invalid LiDAR points. Therefore, to fully utilize the LiDAR scans points and simultaneously limit the network size, we use a VAE to first encode the LiDAR scans into a multivariate Gaussian latent space with mean $\boldsymbol{\mu}_{\text{vae}}$ and variance $\boldsymbol{\sigma}^2_{\text{vae}}$.

The encoder h^{encode}_{vae} of the VAE has one-layer MLP with ReLU that is connected to the input, and two separate one-layer MLPs for encoding $\boldsymbol{\mu}_{\text{vae}}$ and $\boldsymbol{\sigma}^2_{\text{vae}}$. Then, the encoder outputs by random sampling the encoded distribution:

$$\hat{\mathbf{y}} \sim \mathcal{N}(\boldsymbol{\mu}_{\text{vae}}, \boldsymbol{\sigma}^2_{\text{vae}}) \tag{5}$$

The decoder h^{decode}_{vae} of the VAE has two layers of MLP. The first one is with ReLU and the second is with Sigmoid.

D. Optimization

The guaranteed invertibility of INN means that we can do bi-directional training by optimizing loss from both sides of the network. We train both the forward and reverse paths with supervised losses. In each epoch, the forward and reverse paths are both calculated and gradients are added together before an optimizer step.

The VAE network is responsible for encoding and reconstructing the LiDAR scans:

$$\begin{aligned}\hat{\mathbf{y}}_{\text{vae}} &= h^{encode}_{\text{vae}}(\mathbf{y}_{\text{gt}}), \\ \mathbf{y}_{\text{vae}} &= h^{decode}_{\text{vae}}(\hat{\mathbf{y}}_{\text{vae}}),\end{aligned} \tag{6}$$

which is optimized for the commonly used ELBO loss:

$$\mathcal{L}_{\text{vae}} = \|\mathbf{y}_{\text{gt}} - \mathbf{y}_{\text{vae}}\|_1 + \lambda_{\text{KL}}\text{KL}(\mathcal{N}(\boldsymbol{\mu}_{\text{vae}}, \boldsymbol{\sigma}^2_{\text{vae}}), \mathcal{N}(0, 1)), \tag{7}$$

where λ_{KL} is a weight for the KL divergence term. The VAE is trained together with the INN.

Each epoch of training the INN starts with evaluating the encoder of the VAE with ground truth LiDAR scans $\mathbf{y}_{\text{gt}}$ to get the encoded scans $\hat{\mathbf{y}}_{\text{vae}}$, and evaluating the decoder of the VAE with the output of the encoder, as in (6). $\mathcal{L}_{\text{vae}}$ is calculated as in (7). The next step is to evaluate the forward path of the INN with $\hat{\mathbf{x}}_{\text{gt}}$ to get the forward output:

$$[\hat{\mathbf{y}}_{\text{inn}}, \mathbf{z}_{\text{inn}}] = h_{\text{inn}}^{forward}(\hat{\mathbf{x}}_{\text{gt}}, \hat{\mathbf{c}}). \tag{8}$$

We then evaluate the decoder of VAE again with output from the INN forward path:

$$\mathbf{y}_{\text{inn}} = h_{\text{vae}}^{decode}(\hat{\mathbf{y}}_{\text{inn}}), \tag{9}$$

and calculate a loss on the LiDAR scan output:

$$\mathcal{L}_{\mathbf{y}} = \|\mathbf{y}_{\text{gt}} - \mathbf{y}_{\text{inn}}\|_1. \tag{10}$$

We also calculate a loss that matches the output of the INN forward path with the output of the VAE encoder:

$$\mathcal{L}_{\hat{\mathbf{y}}} = \|\hat{\mathbf{y}}_{\text{vae}} - \hat{\mathbf{y}}_{\text{inn}}\|_1. \tag{11}$$

For the reverse path of the INN, we first evaluate with the encoded scan from VAE encoder concatenated by the latent vector generated by the forward path: $[\hat{\mathbf{y}}_{\text{vae}}, \mathbf{z}_{\text{inn}}]$. This produces a predicted pose we call $\hat{\mathbf{x}}_{\text{inn},0}$:

$$\hat{\mathbf{x}}_{\text{inn},0} = h_{\text{inn}}^{reverse}([\hat{\mathbf{y}}_{\text{vae}}, \mathbf{z}_{\text{inn}}], \hat{\mathbf{c}}). \tag{12}$$

We calculated a L1 loss between $\hat{\mathbf{x}}_{\text{inn},0}$ and the ground truth:

$$\mathcal{L}_{\hat{\mathbf{x}},0} = \|\hat{\mathbf{x}}_{\text{gt}} - \hat{\mathbf{x}}_{\text{inn},0}\|_1. \tag{13}$$

Following [15], the intuition of this reverse evaluation is to link the encoded scans plus the predicted latent vector to the single corresponding pose in the ambiguous inverse problem.

To capture the full posterior, we then sample m latent vectors $\mathbf{z} \sim \mathcal{N}(0, 1)$ and evaluate the reverse path using the sampled latent vectors combined with $\hat{\mathbf{y}}_{\text{vae}}$.

$$\hat{\mathbf{x}}_{\text{inn},i} = h_{\text{inn}}^{reverse}([\hat{\mathbf{y}}_{\text{vae}}, \mathbf{z}_i], \hat{\mathbf{c}}), \text{for } i = 1 \ldots m. \tag{14}$$

This generates m poses and we select the minimum of the L1 losses as the second part of the reverse loss:

$$\mathcal{L}_{\hat{\mathbf{x}},i} = \min_{i=1 \ldots m} \|\hat{\mathbf{x}}_{\text{gt}} - \hat{\mathbf{x}}_{\text{inn},i}\|_1. \tag{15}$$

Overall, the training loss of the whole network is:

$$\mathcal{L}_{\text{all}} = \mathcal{L}_{\text{vae}} + \mathcal{L}_{\mathbf{y}} + \lambda_{\hat{\mathbf{y}}}\mathcal{L}_{\hat{\mathbf{y}}} + \mathcal{L}_{\hat{\mathbf{x}},0} + \mathcal{L}_{\hat{\mathbf{x}},i}, \tag{16}$$

where $\lambda_{\hat{\mathbf{y}}}$ is the weight for $\mathcal{L}_{\hat{\mathbf{y}}}$.

IV. Experiments

A. 2D LiDAR Localization on Real-world Robot

We first validate the proposed method of localization with three different 2D LiDAR maps. The first map is a race track in simulation, and the second and third maps are real-world indoor hallway and outdoor environments mapped using the ROS SLAM toolbox with an F1TENTH racecar [38], which is a 1/10 scale autonomous racing car equipped with a Hokuyo 30LX LiDAR and a NVIDIA Jetson Xavier NX board. To collect training data, we uniformly sample $[x, y, \theta]$ on the drivable surface of each map, and use a 2D LiDAR simulator to find the corresponding LiDAR ranges. This means the trained network will be able to localize everywhere on the map. We collect 100k data pairs and train a separate network for each map.

To test the localization performance, we localize a car robot following a test trajectory in each environment and compare the inferred pose with the ground truth. For the real maps, we train with simulated data but test using real LiDAR data on the F1TENTH car driving in indoor and outdoor environments. We approximate the ground truth poses using a particle filter with the full LiDAR inputs and running it offline on a desktop with an infinite compute budget. For a baseline, we configured a GPU-accelerated particle filter [39], so that it can run around the same frequency as the Local_INN on the Jetson NX.

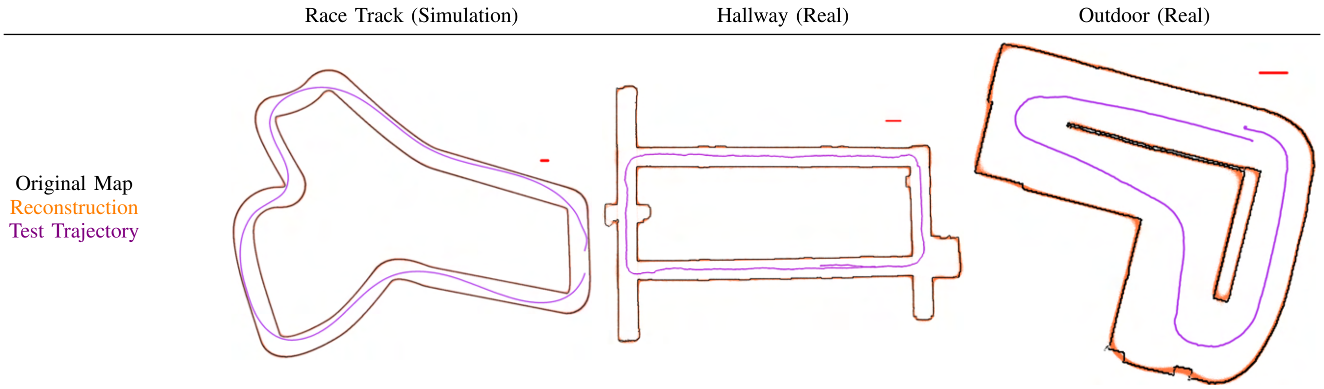
In these experiments, we use 270 points for each LiDAR scan $\mathbf{y}$ covering 270 degrees in front of the LiDAR, following the FoV of the Hokuyo LiDARs. $\hat{\mathbf{y}}$ is set to have 54 dimensions and $\mathbf{z}$ to have 6 dimensions. The encoder of the VAE has one layer of MLP before regressing the $\boldsymbol{\mu}_{\text{vae}}$ and $\boldsymbol{\sigma}_{\text{vae}}^2$ with separate MLP layers. The decoder has two layers of MLP converting 54-dimension $\hat{\mathbf{y}}_{\text{vae}}$ back to 270 ranges points. The INN network has 6 coupling layers, each having separate MLP layers for scale and translation coefficients. We trained the network with batchsize of 500 and with a learning rate that starts from 1×10^{-3} and exponentially decays to 5×10^{-5} in 600 epochs.

The map reconstruction is qualitatively evaluated by calculating the forward path with additionally random sampled test poses. The inferred LiDAR ranges are then converted into the map frame and accumulated to produce an occupancy map. The orange dots in table I are reconstructed maps. We can see the reconstructed map largely overlaps with the real map with some losses in high spatial-frequency information at hard corners. The red bar on the upper right corner of each map is an indicator for 1 meter.

During the inference of the reverse path, we sample latent vector $\mathbf{z}$ and calculate a batch of inferred poses. We can use the covariance of each batch as the confidence of the network. To demonstrate this, we use an Extended Kalman Filter to fuse the network outputs with vehicle odometry. The EKF uses a kinematic bicycle model as the motion model, and the pose output and covariance from the INN as the observation model.

Table I presents localization absolute mean and RMS errors in each environment. We see that not only the localization performance is comparable to particle filter, but the error and RMS also do not increase with vehicle speed. On the contrary, we see the error increase with the particle filter. This is because Local_INN does not directly rely on the smoothness of the state's history, but only relies on the zoning provided by the previous state. Table II compares the runtime of the Local_INN with the GPU-accelerated particle filter we used. We are comparing the latency of Local_INN and particle filter. Other latencies are not accounted for. With runtime optimizations like TensorRT, Local_INN can output localization results with much lower latency than particle filter with almost no decrease in performance, which is crucial in latency-sensitive applications such as high-speed racing [40].

TABLE I
MAP RECONSTRUCTION AND LOCALIZATION ERRORS WITH 2D LIDAR



	Race Track (Simulation)		Hallway (Real)		Outdoor (Real)	
	xy(m)	$\theta(^\circ)$	xy(m)	$\theta(^\circ)$	xy(m)	$\theta(^\circ)$
Online PF (1m/s)	$0.045 \pm \mathbf{0.058}$	0.400 ± 0.512	$\mathbf{0.039} \pm \mathbf{0.066}$	$\mathbf{0.482} \pm 0.808$	$\mathbf{0.013} \pm \mathbf{0.018}$	$\mathbf{0.358} \pm \mathbf{0.456}$
Local_INN (1m/s)	0.050 ± 0.102	0.201 ± 0.532	0.196 ± 0.433	$0.528 \pm \mathbf{0.792}$	0.034 ± 0.047	0.924 ± 1.130
↑ + EKF	$\mathbf{0.039} \pm 0.077$	0.182 ± 0.464	0.093 ± 0.139	0.536 ± 0.797	0.034 ± 0.047	0.917 ± 1.129
↑ + TensorRT	0.039 ± 0.076	$\mathbf{0.177} \pm \mathbf{0.443}$	0.104 ± 0.159	0.547 ± 0.802	0.033 ± 0.046	0.930 ± 1.142
Online PF (5m/s)	0.139 ± 0.168	1.463 ± 2.107	$\mathbf{0.071} \pm \mathbf{0.117}$	0.943 ± 1.738	0.033 ± 0.047	0.940 ± 1.371
Local_INN+EKF (5m/s)	$\mathbf{0.034} \pm \mathbf{0.056}$	$\mathbf{0.133} \pm \mathbf{0.284}$	0.100 ± 0.147	$\mathbf{0.565} \pm \mathbf{0.900}$	$\mathbf{0.032} \pm \mathbf{0.046}$	$\mathbf{0.915} \pm \mathbf{1.130}$

TABLE II
RUNTIME COMPARISONS ON NVIDIA JETSON NX

Online PF	45 Hz
Local_INN (Pytorch)	48 Hz
Local_INN+TensorRT	**270** Hz

B. 3D Open Space LiDAR Localization

We then extended our experiments to using 3D LiDAR data, for which we also have three different environments: Town 10 in the CARLA simulator [41], KAIST in Mulran dataset [42], and Columbia Park in Apollo dataset [22]. For CARLA, we used the simulator to sample all drivable surfaces in the town. To fully train the Local_INN, simulating a large amount of data from the map is preferred. But for comparison with existing works, we just used provided data points for Mulran and Apollo datasets. When testing the network, we report numbers from in-session and out-session localization. For in-session results, in CARLA, we have additionally sampled points; in Mulran and Apollo, we randomly picked and set aside 20% of the dataset for testing. For out-session tests, the network is tested with sequences that are captured at another date. We provide in-session performances to show that the network is able to interpolate between the training data.

We treat 3D LiDAR scans as range images for the 3D experiments. To correctly reconstruct the out-of-range LiDAR points, we added a mask layer to the range images, and an L2 loss on it. The structure and dimension of the network are mostly unchanged for the 3D experiments. The only additions are 6 layers of 2D convolution and transpose convolution layers to the encoder and decoder of the VAE for the range images.

The quality of the map reconstruction is again qualitatively examined as some examples are shown in Fig. 3. Because we simulated many more data points from the CARLA environment, we can see the reconstruction is very close to the original point cloud.

Table III shows a comparison of RMS errors between our method and existing works in localization experiments with 3D LiDARs. Due to the simplicity of our 3D setup, we are comparing to a method from Chen et al [19] that only uses range images from 3D LiDARs, and a method from Yin et al [43] that also uses a neural network with convolution layers to treat LiDAR information. We can see that our results on par with the state-of-the-art.

TABLE III
COMPARISON OF LOCALIZATION RMS ERRORS WITH 3D LIDAR

Methods (xy[m], $\theta[^\circ]$)	CARLA	Mulran	Apollo
Local_INN in-session	$0.27, 0.12$	$0.29, 0.24$	$0.50, 0.26$
Local_INN out-session	$-, -$	$1.41, 1.00$	$1.22, 0.53$
Chen et al.	$0.48, 3.87$	$0.83, 3.14$	$0.57, 3.40$
RaLL (Yin et al.)	$-, -$	$1.27, 1.50$	$-, -$

C. Global Localization

Global localization is needed when a robot starts with an unknown pose or when the robot encounters the kidnapping problem. MCL algorithms usually do global localization by spreading the covariance all around the map and using iterations of control inputs and measurements to decrease the covariance. For Local_INN, the global localization process mainly involves simultaneously tracking multiple assumptions of zoning on the map and a selection process to narrow down the assumptions.

Algorithm 1 shows our global localization process. We track a set $\mathcal{C}$ of n conditional inputs, each with a weight w_i for $i = 1 \dots n$. The set $\mathcal{C}$ is initialized by randomly sample N states in the state space $\mathcal{S}$. We set the total number of latent vectors $\mathbf{z}$ sampled from normal distribution as nM for a constant M. Initially, every $\mathbf{c}_i \in \mathcal{C}$ has the same weight w_i, so each one gets M samples of $\mathbf{z}$.

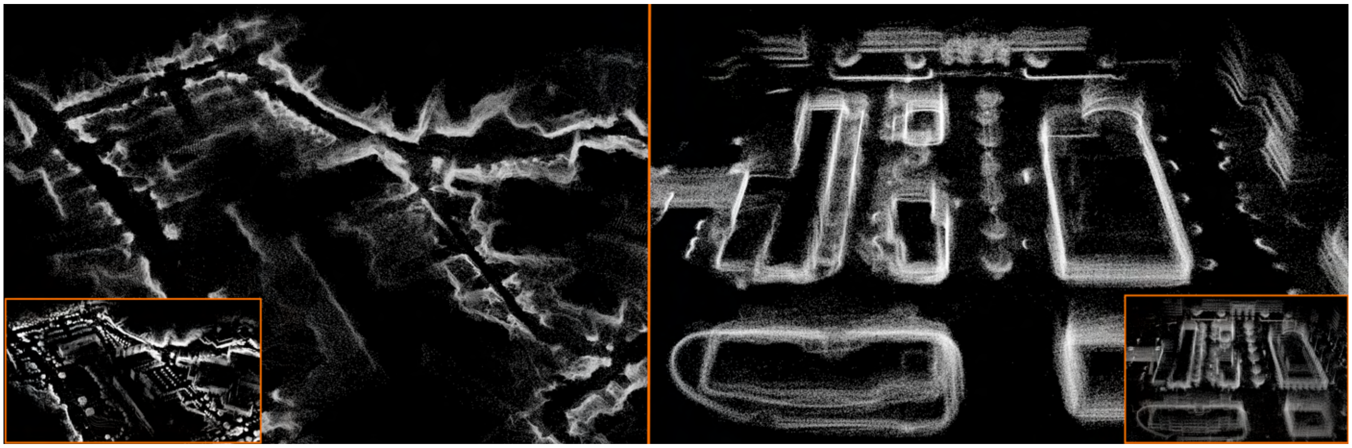


Fig. 3. 3D Map Reconstruction for Mulran and CARLA. Orange boxed thumbnails are original maps. Reconstructions are produced by evaluating the forward path of the Local_INN with poses exterior to the training set.

Algorithm 1 Local_INN Global Localization

$n \leftarrow N, m_i \leftarrow M, w_i \leftarrow 1/M$ for $i = 1 \dots n$
$\mathcal{X}^{\text{rand}} \leftarrow$ random_sample$(\mathcal{S}, n)$
$\mathcal{C}_0 \leftarrow$ convert_to_cond_inputs$(\mathcal{X}^{\text{rand}})$
while *new LiDAR scan* $\mathbf{y}_{t+1}$ *coming* **do**
  for $i = 1 \dots n_t$ **do**
    $\mathbf{x}_{t+1,i} \leftarrow$ Local_INN_reverse$(\mathbf{y}_{t+1}, \mathbf{c}_i, m_i)$
    $\mathcal{X}_{t+1}$.append$(\mathbf{x}_{t+1,i})$
    $\mathbf{y}_{\text{inn},i} \leftarrow$ Local_INN_forward$(\mathbf{x}_{t+1,i}, \mathbf{c}_i)$
    $w_i \leftarrow 1/\|\mathbf{y}_{\text{inn},i} - \mathbf{y}_{t+1}\|_1$
  end for
  $\mathcal{C}_{t+1} \leftarrow$ convert_to_cond_inputs$(\mathcal{X}_{t+1})$
  $n_{t+1} \leftarrow |\mathcal{C}_{t+1}|$
  for $i = 1 \dots n_{t+1}$ **do**
    $m_i \leftarrow$ normalized$(w_i) n_{t+1} M$
  end for
end while

When a new LiDAR scan arrives, for each $\mathbf{c}_i \in \mathcal{C}$, we evaluate the reverse path of the Local_INN with m_i samples of latent vector $\mathbf{z}$. The output poses from Local_INN become the next $\mathcal{C}$. We then update the weight w_i for every $\mathbf{c}_i$ using the reciprocal of the scan error, calculated with the current sensor measurement, and inferred LiDAR scan from evaluating the forward path of the Local_INN. We favor the $\mathbf{c}_i \in \mathcal{C}$ that have higher weights by redistributing $\mathbf{z}$ samples based on the weights. Those with higher weights will have more $\mathbf{z}$ samples, which in turn may result in better pose estimations. It also should be noted that the size of $\mathcal{C}$ will decrease as iterations go because repeated elements in $\mathcal{C}$ are combined. Hence, we design a selection process to find the best-fit candidate. Lastly, we record the accumulated weights for every iteration and the $\mathbf{c}_i$ with the highest accumulated weights will be the most likely zone that the robot exists in.

We test out the above algorithm with different environments. We define a `Converged` as the correct pose having the highest weight and a `Tracking` as the correct pose within the top 5 on the tracking list. Table IV presents the percentage of `Converged` cases, `Tracking` cases, and the absolute mean errors if the correct pose is picked or in tracking at the 10th iteration. The starting poses are randomly picked and the rates are out of 2k tests in each environment. We use the test trajectory for the 2D maps and out-of-session test sets for the 3D maps. The result shows the neural network can quickly identify correct poses with only 10 LiDAR scans. We can also see in the Hallway map, that the convergence of the assumptions is slower, which is expected in this highly symmetrical environment. As the algorithm keeps iterating with new LiDAR data, it will eventually converge to the correct pose.

TABLE IV
GLOBAL LOCALIZATION SUCCESS RATES IN DIFFERENT ENVIRONMENTS AT ITERATION 10

Map	Converged	Tracking	Δ_{xy}, Δ_θ
Race Track	79.5%	99.5%	0.075, 0.274
Hallway	66.4%	91.1%	0.258, 0.538
Outdoor	98.5%	100%	0.049, 0.911
Mulran	93.5%	95.0%	0.884, 0.454
Apollo	82.5%	83.0%	1.569, 0.122

V. CONCLUSION

In this paper, we present a normalizing flow-based framework to solve the robot localization problem. The trained INN provides a bijective mapping between map information and robot poses. While localizing, sampling the latent space gives us a mean and covariance, which can be used as uncertainty estimation for the fusing with other data sources. In our 2D experiments, Local_INN is on par with particle filer on accuracy by providing localization with errors as low as 0.032 m and 0.915°, while much fast by running 270Hz on an embedded platform. Such low latency combined with the fact that its error does not significantly increase with robot velocity makes it suitable for high-speed applications. We also show that Local_INN has great potential in 3D LiDAR localization with errors of 0.29 m, 0.24° in-session, and 1.41 m, 1.00° out-of-session on the Mulran dataset. Moreover, with our global localization algorithm, Local_INN has a convergence rate of 93.5% in the Mulran dataset at the 10th iteration.

References

[1] F. Dellaert, D. Fox, W. Burgard, and S. Thrun, "Monte carlo localization for mobile robots," in *Proceedings 1999 IEEE international conference on robotics and automation (Cat. No. 99CH36288C)*, vol. 2. IEEE, 1999, pp. 1322–1328.

[2] S. Thrun, "Probabilistic robotics," *Communications of the ACM*, vol. 45, no. 3, pp. 52–57, 2002.

[3] V. Fox, J. Hightower, L. Liao, D. Schulz, and G. Borriello, "Bayesian filtering for location estimation," *IEEE pervasive computing*, vol. 2, no. 3, pp. 24–33, 2003.

[4] R. Dube, A. Cramariuc, D. Dugas, H. Sommer, M. Dymczyk, J. Nieto, R. Siegwart, and C. Cadena, "Segmap: Segment-based mapping and localization using data-driven descriptors," *The International Journal of Robotics Research*, vol. 39, no. 2-3, pp. 339–355, 2020.

[5] P.-E. Sarlin, C. Cadena, R. Siegwart, and M. Dymczyk, "From coarse to fine: Robust hierarchical localization at large scale," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2019, pp. 12 716–12 725.

[6] E. G. Tabak and E. Vanden-Eijnden, "Density estimation by dual ascent of the log-likelihood," *Communications in Mathematical Sciences*, vol. 8, no. 1, pp. 217–233, 2010.

[7] L. Dinh, J. Sohl-Dickstein, and S. Bengio, "Density estimation using real nvp," *arXiv preprint arXiv:1605.08803*, 2016.

[8] D. P. Kingma and P. Dhariwal, "Glow: Generative flow with invertible 1x1 convolutions," *Advances in neural information processing systems*, vol. 31, 2018.

[9] G. Papamakarios, E. T. Nalisnick, D. J. Rezende, S. Mohamed, and B. Lakshminarayanan, "Normalizing flows for probabilistic modeling and inference." *J. Mach. Learn. Res.*, vol. 22, no. 57, pp. 1–64, 2021.

[10] L. Ardizzone, J. Kruse, S. Wirkert, D. Rahner, E. W. Pellegrini, R. S. Klessen, L. Maier-Hein, C. Rother, and U. Köthe, "Analyzing inverse problems with invertible neural networks," *arXiv preprint arXiv:1808.04730*, 2018.

[11] L. Ardizzone, C. Lüth, J. Kruse, C. Rother, and U. Köthe, "Guided image generation with conditional invertible neural networks," *arXiv preprint arXiv:1907.02392*, 2019.

[12] T. J. Adler, L. Ardizzone, A. Vemuri, L. Ayala, J. Gröhl, T. Kirchner, S. Wirkert, J. Kruse, C. Rother, U. Köthe *et al.*, "Uncertainty-aware performance assessment of optical imaging modalities with invertible neural networks," *International journal of computer assisted radiology and surgery*, vol. 14, no. 6, pp. 997–1007, 2019.

[13] M. Xiao, S. Zheng, C. Liu, Y. Wang, D. He, G. Ke, J. Bian, Z. Lin, and T.-Y. Liu, "Invertible image rescaling," in *European Conference on Computer Vision*. Springer, 2020, pp. 126–144.

[14] R. Zhao, T. Liu, J. Xiao, D. P. Lun, and K.-M. Lam, "Invertible image decolorization," *IEEE Transactions on Image Processing*, vol. 30, pp. 6081–6095, 2021.

[15] T. Wehrbein, M. Rudolph, B. Rosenhahn, and B. Wandt, "Probabilistic monocular 3d human pose estimation with normalizing flows," in *International Conference on Computer Vision (ICCV)*, Oct. 2021.

[16] D. P. Kingma and M. Welling, "Auto-encoding variational bayes," *arXiv preprint arXiv:1312.6114*, 2013.

[17] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin, "Attention is all you need," *Advances in neural information processing systems*, vol. 30, 2017.

[18] C. Zhang, M. H. Ang, and D. Rus, "Robust lidar localization for autonomous driving in rain," in *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2018, pp. 3409–3415.

[19] X. Chen, I. Vizzo, T. Läbe, J. Behley, and C. Stachniss, "Range Image-based LiDAR Localization for Autonomous Vehicles," in *Proc. of the IEEE Intl. Conf. on Robotics & Automation (ICRA)*, 2021.

[20] X. Chen, T. Läbe, L. Nardi, J. Behley, and C. Stachniss, "Learning an overlap-based observation model for 3d lidar localization," in *2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2020, pp. 4602–4608.

[21] I. A. Barsan, S. Wang, A. Pokrovsky, and R. Urtasun, "Learning to localize using a lidar intensity map," *arXiv preprint arXiv:2012.10902*, 2020.

[22] W. Lu, Y. Zhou, G. Wan, S. Hou, and S. Song, "L3-net: Towards learning based lidar localization for autonomous driving," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2019, pp. 6389–6398.

[23] R. Clark, S. Wang, A. Markham, N. Trigoni, and H. Wen, "Vidloc: A deep spatio-temporal model for 6-dof video-clip relocalization," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2017, pp. 6856–6864.

[24] M. A. Uy and G. H. Lee, "Pointnetvlad: Deep point cloud based retrieval for large-scale place recognition," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2018, pp. 4470–4479.

[25] Y. Cho, G. Kim, S. Lee, and J.-H. Ryu, "Openstreetmap-based lidar global localization in urban environment without a prior lidar map," *IEEE Robotics and Automation Letters*, vol. 7, no. 2, pp. 4999–5006, 2022.

[26] L. Sun, D. Adolfsson, M. Magnusson, H. Andreasson, I. Posner, and T. Duckett, "Localising faster: Efficient and precise lidar-based robot localisation in large-scale environments," in *2020 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2020, pp. 4386–4392.

[27] M. Cai, C. Shen, and I. Reid, "A hybrid probabilistic model for camera relocalization," 2019.

[28] A. Kendall and R. Cipolla, "Modelling uncertainty in deep learning for camera relocalization," in *2016 IEEE international conference on Robotics and Automation (ICRA)*. IEEE, 2016, pp. 4762–4769.

[29] H. Deng, M. Bui, N. Navab, L. Guibas, S. Ilic, and T. Birdal, "Deep bingham networks: Dealing with uncertainty and ambiguity in pose estimation," *International Journal of Computer Vision*, pp. 1–28, 2022.

[30] X. Wei, I. A. Bârsan, S. Wang, J. Martinez, and R. Urtasun, "Learning to localize through compressed binary maps," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2019, pp. 10 316–10 324.

[31] B. Mildenhall, P. P. Srinivasan, M. Tancik, J. T. Barron, R. Ramamoorthi, and R. Ng, "Nerf: Representing scenes as neural radiance fields for view synthesis," *Communications of the ACM*, vol. 65, no. 1, pp. 99–106, 2021.

[32] E. Sucar, S. Liu, J. Ortiz, and A. J. Davison, "imap: Implicit mapping and positioning in real-time," in *Proceedings of the IEEE/CVF International Conference on Computer Vision*, 2021, pp. 6229–6238.

[33] Z. Zhu, S. Peng, V. Larsson, W. Xu, H. Bao, Z. Cui, M. R. Oswald, and M. Pollefeys, "Nice-slam: Neural implicit scalable encoding for slam," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2022, pp. 12 786–12 796.

[34] C. Durkan, A. Bekasov, I. Murray, and G. Papamakarios, "Neural spline flows," *Advances in neural information processing systems*, vol. 32, 2019.

[35] H. Wu, J. Köhler, and F. Noé, "Stochastic normalizing flows," *Advances in Neural Information Processing Systems*, vol. 33, pp. 5933–5944, 2020.

[36] Q. Zhang and Y. Chen, "Diffusion normalizing flow," *Advances in Neural Information Processing Systems*, vol. 34, pp. 16 280–16 291, 2021.

[37] C. Winkler, D. Worrall, E. Hoogeboom, and M. Welling, "Learning likelihoods with conditional normalizing flows," *arXiv preprint arXiv:1912.00042*, 2019.

[38] M. O'Kelly, H. Zheng, D. Karthik, and R. Mangharam, "F1tenth: An open-source evaluation environment for continuous control and reinforcement learning," in *Proceedings of the NeurIPS 2019 Competition and Demonstration Track*, ser. Proceedings of Machine Learning Research, H. J. Escalante and R. Hadsell, Eds., vol. 123. PMLR, 08–14 Dec 2020, pp. 77–89.

[39] C. Walsh and S. Karaman, "Cddt: Fast approximate 2d ray casting for accelerated localization," vol. abs/1705.01167, 2017. [Online]. Available: http://arxiv.org/abs/1705.01167

[40] J. Betz, H. Zheng, A. Liniger, U. Rosolia, P. Karle, M. Behl, V. Krovi, and R. Mangharam, "Autonomous Vehicles on the Edge: A Survey on Autonomous Vehicle Racing," *IEEE Open Journal of Intelligent Transportation Systems*, vol. 3, pp. 458–488, 2022.

[41] A. Dosovitskiy, G. Ros, F. Codevilla, A. Lopez, and V. Koltun, "Carla: An open urban driving simulator," in *Conference on robot learning*. PMLR, 2017, pp. 1–16.

[42] G. Kim, Y. S. Park, Y. Cho, J. Jeong, and A. Kim, "Mulran: Multimodal range dataset for urban place recognition," in *2020 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2020, pp. 6246–6253.

[43] H. Yin, R. Chen, Y. Wang, and R. Xiong, "Rall: end-to-end radar localization on lidar map using differentiable measurement model," *IEEE Transactions on Intelligent Transportation Systems*, 2021.

PoseINN: Realtime Visual-based Pose Regression and Localization with Invertible Neural Networks

Zirui Zang, Ahmad Amine, Rahul Mangharam

***Abstract*—Estimating ego-pose from cameras is an important problem in robotics with applications ranging from mobile robotics to augmented reality. While SOTA models are becoming increasingly accurate, they can still be unwieldy due to high computational costs. In this paper, we propose to solve the problem by using invertible neural networks (INN) to find the mapping between the latent space of images and poses for a given scene. Our model achieves similar performance to the SOTA while being faster to train and only requiring offline rendering of low-resolution synthetic data. By using normalizing flows, the proposed method also provides uncertainty estimation for the output. We also demonstrated the efficiency of this method by deploying the model on a mobile robot.**

I. Introduction

Visual pose regression is the task of finding camera poses of images within a trained environment. The matured geometric-based pipeline [1]–[3] can lead to expensive computation and long latency. On the other hand, learning-based pose regression [4]–[8] has improved efficiency but can be cumbersome to deploy due to their low accuracy and long training time. Recently, with aid from neural radiance fields (NeRF) [9], learning-based pose regression methods have greatly improved their accuracy [10], [11]. Direct feature-matching with online-rendered images and synthetic training data generation are two ways people use NeRF to improve pose regression. Despite that, these efforts either need online rendering with NeRF or long-time synthetic data preparation.

To address these limitations, we propose to use NeRF to render a large number of low-resolution images and view the problem as finding a mapping between the distributions of camera poses and images with normalizing flows. NeRF enabled us to conveniently sample in the image space and fully utilize the 3D spatial information embedded in the training dataset. During the evaluation, we can find the full posterior distribution of poses given the images by sampling the latent space of the INN. We summarize our contributions as the following:

1) We extend Local_INN [12] from LiDAR to cameras, which expands the usability for real robots. The method is tested on common benchmark datasets and the performance is on par with state-of-the-art.
2) We realize a fast data preparation pipeline with NeRF [9], [13], which further lowers the deployment burden.
3) We demonstrate the balance of performance and efficiency of the proposed method by deploying it on a real mobile robot.

All authors are with the University of Pennsylvania, Department of Electrical and Systems Engineering, 19104, Philadelphia, PA, USA. Emails: {zzang, aminea, rahulm}@seas.upenn.edu

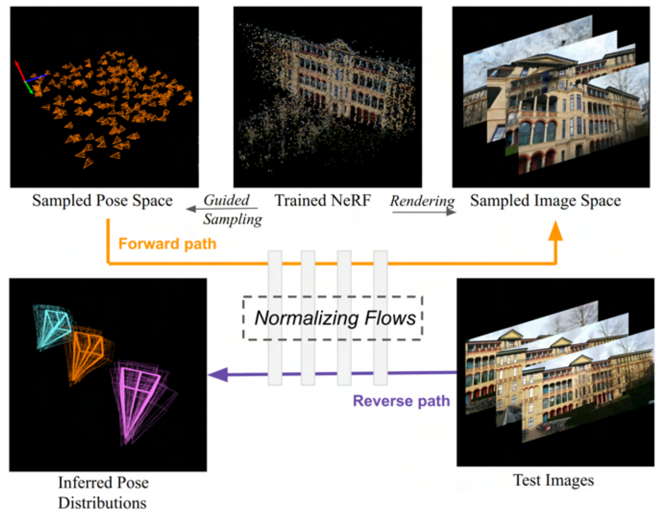


Fig. 1. We propose to learn a mapping between the latent space of the images and camera poses in an environment with an invertible neural network. We use NeRF to guide camera pose sampling and render synthetic images. Evaluating the reverse path of the INN outputs the full posterior distribution of camera poses given a test image.

II. Related Work

A. Visual Pose Regression

The pioneering work in pose regression by PoseNet [4] used simple CNN + average pooling layers to regress camera poses. Since that, the state-of-the-art (SOTA) was yearly refreshed by people trying more complex neural network architectures, such as using separate outputs for position and orientation [5], translational invariant layers [6], [8], or LSTM [14], auto-encoders [15], transformers [16], etc. To show the effectiveness of our method, we are using an encoder that is also simply CNN + average pooling and yet still performs on par with the SOTAs.

Recently, pose regression tasks benefited from NeRF's ability to render photo-realistic images from novel camera poses. LENS [10] augments the training data by rendering synthetic images with a trained NeRF-W from a grid-based novel pose sampling. The limitation of LENS is the days-long training time and high-resolution image rendering time. On the other hand, Direct-PoseNet [17] uses a photometric loss to compare the test images with NeRF-rendered images at test poses. DFNet [11] improved that with direct feature-matching in the feature space instead of pixel-value space. However, these methods require expensive online rendering from NeRF. Different from the SOTA's complex approach, we claim that offline rendering of many low-resolution images is enough to perform the localization. Given a test image, our method also produces the posterior distribution of camera poses, which can be used as uncertainty estimations [6], [18], [19] to improve robustness and deployability.

B. Normalizing Flows

Normalizing flows use a series of bijective transformations to map a source distribution to a target distribution. They provide efficient density estimation [20], [21] and sampling of the target distribution. Ardizonne et al. [22]–[24] proposed a framework for using normalizing flows to solve ambiguous inverse problems. The use of INNs in solving inverse problems has been applied to various fields [22], [25]–[27]. Recently, Local_INN [12] has shown the effectiveness of INNs in performing robot localization, which is naturally an ambiguous inverse problem. However, [12] uses LiDAR ranges, which can be simulated with high fidelity given an occupancy map of the environment. Although LiDAR data provide reliable distance measurements, the sensor is expensive and lacks color information about the world. We extend that framework for visual 6DoF pose regression which is a more common problem. For that we developed a synthetic pose sampling policy with NeRF guidance.

III. Method

Our approach to visual pose regression is to view it as finding a mapping from the distribution of the image to that of camera poses. Effectively sampling enough corresponding data points in both distributions is the key to finding such mapping. We train a Neural Radiance Fields (NeRF) model of the environment and use it to render images at randomly sampled novel camera poses as in Fig. 2. We propose a random camera pose sampling and synthetic image rendering pipeline that is fundamental to the final pose regression result.

Once we have generated enough image samples, based on [12], we use normalizing flows combined with variational autoencoder (VAE) to learn the mapping from pose to images. To reduce the dimensionality of image data, we use a VAE to encode images into a latent space. Then, we use coupling-based normalizing flows to learn the mapping from encoded images to poses. The latent space of the normalizing flows is sampled according to a normal distribution during training. During the evaluation, we evaluate only the encoder of the VAE and the reverse path of the normalizing flows with repeatedly sampled INN latent space to reveal the full posterior distribution of the poses given an input image.

A. Generate Synthetic Views with NeRF

A NeRF model stores 3D spatial information of an environment implicitly within two neural networks: A density MLP and a color MLP, which can be queried for any point in the continuous 3D space. Images can be rendered by tracing rays from the environment to the image plane, integrating the density and color information provided by the two MLPs. We train a NeRF model with a set of images with known camera poses, optimizing the rendering loss. However, if the learned density and color information is noisy or missing, the rendered images will contain artifacts or be a complete mess. Therefore, selecting suitable rendering poses while sufficiently sampling the wanted 3D space is challenging.

We used nerfacto [13] as our NeRF model. After training the model, we output a sparse point cloud from NeRF by thresholding the density of the environment. To generate novel camera poses, we first uniformly randomly sample positions in the region. The orientations of these sampled camera poses are given as $R_{\text{noise}} R^{\text{rand}}_{\text{training}}$, where $R^{\text{rand}}_{\text{training}}$ is a randomly picked camera orientation from the training set and R_{noise} is an added perturbation. We generated random rotation R_{noise} for up to 3.6 degrees using [28].

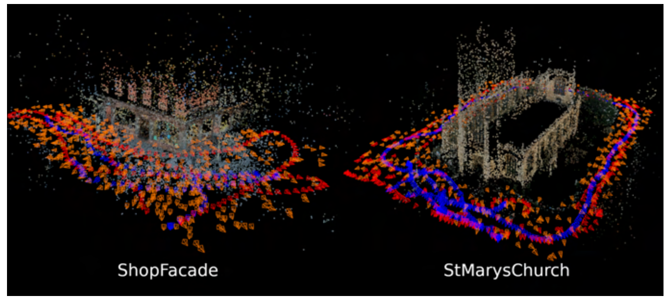


Fig. 2. Sampling of Novel Camera Poses. Point clouds represent high-density points in the environment. Small pyramids represent training poses, testing poses, and sampled poses.

For each sampled camera pose, we verify that we have sufficient spatial information by finding a subset $\mathcal{P}_{\text{in-view}}$ of the NeRF point cloud that is within the field of view (in-view) of the sample camera. We want every sampled camera pose to have enough $\mathcal{P}_{\text{in-view}}$, i.e. enough density information for rendering, and not blocked by a very close point in $\mathcal{P}_{\text{in-view}}$. We then filter out the sampled camera poses according to the following three rules:

- The distance δ_{training} from the sampled pose to the nearest pose in the training set cannot be larger than 0.5 meters.
- For $N_{\text{in-view}} = |\mathcal{P}_{\text{in-view}}|$, we first find the range of $N_{\text{in-view}}$ of the poses in training set. Then we limit the $N_{\text{in-view}}$ of sampled poses according to that range.
- The distance $\delta_{\text{in-view}}$ from the sampled pose to the nearest point in $\mathcal{P}_{\text{in-view}}$ is also limited with the range of $\delta_{\text{in-view}}$ of the poses in training set.

Because we use a sparse point cloud, we can sample 50k poses within minutes. Synthetic images at the sampled poses are then rendered with the trained NeRF model.

B. Learning the Pose-Image Mapping

Normalizing flows are a series of transformations that are mathematically invertible and with learnable parameters. Fig. 3, shows the structure of the network. The normalizing flows side of the network is identical to [12], please refer to that paper for details. We use Real-NVP [20], [21] for its efficiency, which uses affine coupling blocks to achieve invertibility. $\mathbf{c}$ is the optional conditional input [23]. For a fair comparison with other methods, we don't use $\mathbf{c}$ for the absolute pose regression experiments. It's only for real robot localization experiments. Normalizing flows require the input and output to have the same dimension due to their invertibility. The 6DoF camera poses, $\mathbf{x} = [x, y, z, \theta_z, \theta_x, \theta_y]$

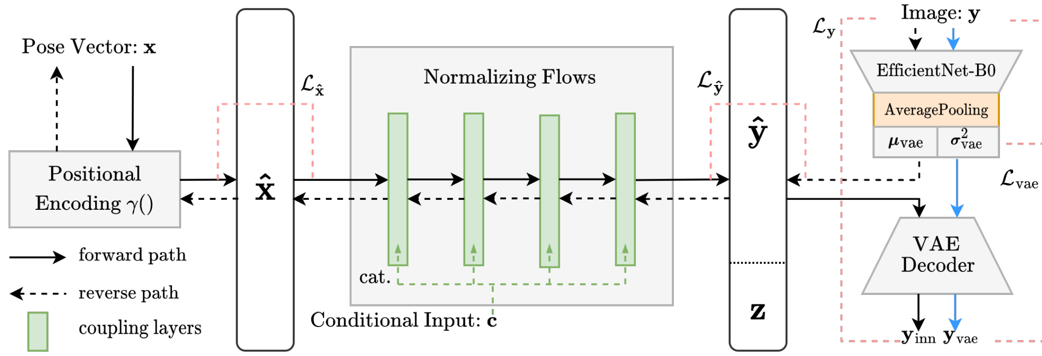


Fig. 3. Network Structure of the PoseINN. The forward path (solid) is from pose to image. The reverse path (dashed) is from image to pose.

are augmented with Positional Encoding [29] [9] from $\mathbb{R}^6$ to $\mathbb{R}^{12L}$.

$$\gamma(p) = (\sin(2^0\pi p), \cos(2^0\pi p), \ldots, \sin(2^{L-1}\pi p), \cos(2^{L-1}\pi p)). \tag{1}$$

We use $L = 5$ for camera poses and the output is concatenated with the original 6-dimensional pose to form an input $\hat{\mathbf{x}} \in \mathbb{R}^{12L+6}$ for the INN. On the image side, we use a VAE to encode the image $\mathbf{y}$ into $\hat{\mathbf{y}} \in \mathbb{R}^{12L}$, which is concatenated with a 6-dimensional latent vector $\mathbf{z} \sim \mathcal{N}(0, 1)$ to form the output of the INN. Different from [12], in the VAE encoder, we use a pre-trained EfficientNet-B0 backbone [30] connected with an average-pooling layer to output one number for each feature channel. At test time, we can sample the latent vector to reveal the full posterior distribution of the pose given an image [23].

We train the network the same way as in [12], where with each batch of data, we evaluate both the forward and reverse paths of the network and losses are added together before an optimizer step. To handle the 6DoF poses more efficiently, we used the geodesic distance [31] $\mathcal{L}_{\text{geo}}$ between two rotations:

$$\mathcal{L}_{\text{geo}} = \cos^{-1}((tr(M_{\text{pred}} M_{\text{gt}}^{-1}) - 1)/2). \tag{2}$$

The EfficientNet backbone in the VAE is loaded with pre-trained weights when initialized and also optimized with the rest of the network in training.

IV. Experiments

We validated our method with two types of tasks. To directly compare it with other pose regression methods, we tested on public absolute pose regression datasets. We also deployed a sequential version on a mobile robot to show the performance of our method on an embedded platform.

A. Camera Pose Regression on Public Dataset

TABLE I
Data Generation Strategy Comparison
(Error Data from 7Scene)

Model (backbone)	Pose Error (m/°)	Synthetic Resolution	Rendering Cost	Generation Mode
LENS(EB3)	0.08/3.00	High	Expensive	Offline
DFNet(EB0)	**0.08**/3.47	Low	Cheap	Online
Ours(EB0)	0.09/**2.65**	**Low**	**Cheap**	**Offline**

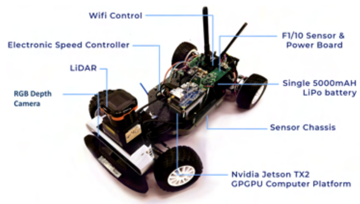
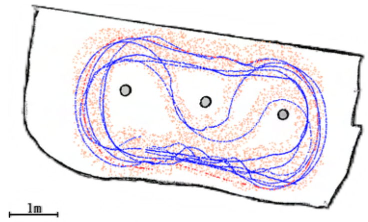
With the 7scene [32] dataset, we trained the nerfacto model for 50k epochs, which takes about 20 mins on our setup with an NVIDIA A6000 GPU. Then 50k synthetic camera-pose images are rendered for each scene, which takes about 40 mins. The rendering resolution for the 7scene dataset is 160x120. The original training set images are then mixed with the rendered images and resized to 128x128 for training the INN. We trained the network for 300 epochs with batch size 200 and a learning rate of 5e-4 exponentially decaying to 5e-5, which takes around 8 hours. Table I shows a comparison of the data generation strategy with LENS [10] and DFNet [11]. The inputs for the other two methods are from [11]. Our strategy is the most efficient while outputting on-par results.

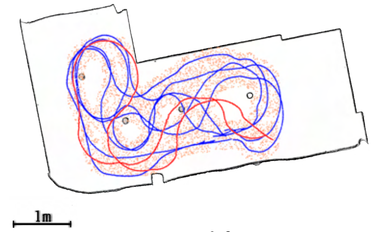
B. Visual Localization on Real-world Mobile Robot

Fig. 4. Examples of training and rendered images in real-world testing (Up: Indoor, Down: Outdoor)

With a small network size, PoseINN is suitable for embedded platforms. To demonstrate that, we deployed PoseINN on an F1TENTH racecar [33], which is a 1/10 scale autonomous racing car equipped with a Hokuyo 30LX LiDAR, an RGB camera, and an NVIDIA Jetson Xavier NX. We used LiDAR to collect ground truth poses for training images and used the camera for localization tests. For the 2D localization experiment, we train for 3 degrees of freedom: xy positions and the car's heading. Similar to [12], the network architecture

TABLE II
MEDIAN LOCALIZATION ERRORS WITH 2D LIDAR VS. CAMERA

<table>
<tr><th>Experiment Platform</th><th></th><th>Indoor</th><th>Outdoor</th></tr>
<tr><td>
</td><td>train trajectory
test trajectory
sampled points</td><td>

(xy[m], θ[°])</td><td>

(xy[m], θ[°])</td></tr>
<tr><td></td><td>Online PF (45Hz)</td><td>$\mathbf{0.01}, 0.23$</td><td>$\mathbf{0.02}, \mathbf{0.36}$</td></tr>
<tr><td></td><td>PoseINN (154Hz)</td><td>$0.02, 0.31$</td><td>$0.12, 0.72$</td></tr>
<tr><td></td><td>PoseINN + EKF</td><td>$0.02, \mathbf{0.22}$</td><td>$0.10, 0.65$</td></tr>
</table>

we used for the 2D localization experiments takes a rounded previous state of the mobile robot as conditional input **c**, which is encoded by a separate MLP. This one-step historical information makes the inverse problem easier and it's used in traditional robot localization methods like particle filters [34], [35].

We set up an indoor and an outdoor experiment. The maps shown in the Table II are captured with LiDAR scan using ROS SLAM toolbox. We use an offline particle filter [36] with an infinite computation budget for ground truth poses and training data for NeRF. An online version of the particle filter with fewer particles is used as the baseline comparison. Training and testing trajectories are also shown on the map. We capture RGB images as the car navigates along the trajectories. Fig. 4 shows the training and rendered images. We can see even without the super-accurate image renderings, the trained model is still able to provide localization.

The translation and rotation error results in Table II show that when the training data sufficiently cover the test trajectory, this method can provide localization comparable to LiDAR-based PF. When the test trajectory moves outside the sampled zone, then the performance drops. As for runtime on the Jetson Xavier NX, PoseINN runs at 154Hz while evaluating batches with 50 randomly sampled **z** for uncertainty estimation, whereas the compared online particle filter runs at 45Hz.

C. Uncertainty Estimation

TABLE III
OUTPUT FILTERING WITH UNCERTAINTY ESTIMATION
(ERROR DATA FROM CAMBRIDGE [4])

(xy[m], θ[°])	Raw Mean Error	With Filtering
Kings	0.93, 1.02	**0.58**, **0.96**
Hospital	0.87, 1.14	**0.64**, **0.97**
Shop	0.59, 5.00	**0.20**, **1.04**
Church	0.81, 2.43	**0.52**, **1.21**
Average	0.84, 2.55	**0.68**, **1.87**

We can then calculate the variance of the output distribution as uncertainty estimations. To demonstrate the effectiveness, we use the covariance of the 2D localization results with an Extended Kalman Filter (EKF) to fuse the output with odometry data from the mobile robot, which improves the accuracy. For the 3D pose regression experiments, we show that filtering the inferred poses with their variance reduces noise levels. In Table III, we show the *average error* of the raw outputs of PoseINN on the left. We then filter out outputs with variance values larger than the median variance value of the testing set. The average errors of outputs after filtering are in the right column. Because average values can be influenced by extreme values, a large improvement shows the output is more robust.

V. DISCUSSION & LIMITATIONS

Using NeRF to efficiently sample camera poses and RGB images in an environment, we reduce the problem of pose regression into learning a mapping between two distributions. Results in Table I show that with a large amount of lower-resolution rendering, we can achieve the same performance without using more complex methods or higher resolutions as in the compared methods. Results in Table II show the proposed method is very efficient and can provide accurate localization if proper training data is provided. The uncertainty estimation that naturally comes with the normalizing flows also makes it suitable for deployment on robot platforms.

Some limitations remain in this work. First, we didn't deal with the domain gap between NeRF-rendered images and the real images that change dramatically with the weather, camera parameters, etc. We tried to have the VAE reconstruct rendered images from real images, but the effect was not prominent. Second, we can see from our experiment that better-covered training data is crucial for the final results. Although our camera pose sampling pipeline, reduced instances of bad renderings, having a more deeply related rendering pipeline will be very helpful.

VI. CONCLUSION

We showcase how this [12] invertible neural network architecture can be used for image-based localization at SOTA performance by only changing an image encoder. To achieve that, we used NeRF as a camera simulator to efficiently sample images within an environment. The efficiency and robustness of the model are illustrated by deploying it on an embedded mobile robot.

References

[1] J. L. Schonberger and J.-M. Frahm, "Structure-from-motion revisited," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 4104–4113.

[2] A. Fisher, R. Cannizzaro, M. Cochrane, C. Nagahawatte, and J. L. Palmer, "Colmap: A memory-efficient occupancy grid mapping framework," *Robotics and Autonomous Systems*, vol. 142, p. 103755, 2021.

[3] T. Sattler, B. Leibe, and L. Kobbelt, "Improving image-based localization by active correspondence search," in *Computer Vision–ECCV 2012: 12th European Conference on Computer Vision, Florence, Italy, October 7-13, 2012, Proceedings, Part I 12*. Springer, 2012, pp. 752–765.

[4] A. Kendall, M. Grimes, and R. Cipolla, "Posenet: A convolutional network for real-time 6-dof camera relocalization," in *Proceedings of the IEEE international conference on computer vision*, 2015, pp. 2938–2946.

[5] J. Wu, L. Ma, and X. Hu, "Delving deeper into convolutional neural networks for camera relocalization," in *2017 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2017, pp. 5644–5651.

[6] A. Moreau, N. Piasco, D. Tsishkou, B. Stanciulescu, and A. de La Fortelle, "Coordinet: uncertainty-aware pose regressor for reliable vehicle localization," in *Proceedings of the IEEE/CVF Winter Conference on Applications of Computer Vision*, 2022, pp. 2229–2238.

[7] I. Melekhov, J. Ylioinas, J. Kannala, and E. Rahtu, "Image-based localization using hourglass networks," in *Proceedings of the IEEE international conference on computer vision workshops*, 2017, pp. 879–886.

[8] R. Liu, J. Lehman, P. Molino, F. Petroski Such, E. Frank, A. Sergeev, and J. Yosinski, "An intriguing failing of convolutional neural networks and the coordconv solution," *Advances in neural information processing systems*, vol. 31, 2018.

[9] B. Mildenhall, P. P. Srinivasan, M. Tancik, J. T. Barron, R. Ramamoorthi, and R. Ng, "Nerf: Representing scenes as neural radiance fields for view synthesis," *Communications of the ACM*, vol. 65, no. 1, pp. 99–106, 2021.

[10] A. Moreau, N. Piasco, D. Tsishkou, B. Stanciulescu, and A. de La Fortelle, "Lens: Localization enhanced by nerf synthesis," in *Conference on Robot Learning*. PMLR, 2022, pp. 1347–1356.

[11] S. Chen, X. Li, Z. Wang, and V. A. Prisacariu, "Dfnet: Enhance absolute pose regression with direct feature matching," in *Computer Vision–ECCV 2022: 17th European Conference, Tel Aviv, Israel, October 23–27, 2022, Proceedings, Part X*. Springer, 2022, pp. 1–17.

[12] Z. Zang, H. Zheng, J. Betz, and R. Mangharam, "Local_inn: Implicit map representation and localization with invertible neural networks," *arXiv preprint arXiv:2209.11925*, 2022.

[13] M. Tancik, E. Weber, E. Ng, R. Li, B. Yi, J. Kerr, T. Wang, A. Kristoffersen, J. Austin, K. Salahi, A. Ahuja, D. McAllister, and A. Kanazawa, "Nerfstudio: A modular framework for neural radiance field development," *arXiv preprint arXiv:2302.04264*, 2023.

[14] B. Wang, C. Chen, C. X. Lu, P. Zhao, N. Trigoni, and A. Markham, "Atloc: Attention guided camera localization," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 34, no. 06, 2020, pp. 10 393–10 401.

[15] Y. Shavit and Y. Keller, "Camera pose auto-encoders for improving pose regression," in *Computer Vision–ECCV 2022: 17th European Conference, Tel Aviv, Israel, October 23–27, 2022, Proceedings, Part X*. Springer, 2022, pp. 140–157.

[16] Y. Shavit, R. Ferens, and Y. Keller, "Learning multi-scene absolute pose regression with transformers," in *Proceedings of the IEEE/CVF International Conference on Computer Vision*, 2021, pp. 2733–2742.

[17] S. Chen, Z. Wang, and V. Prisacariu, "Direct-posenet: absolute pose regression with photometric consistency," in *2021 International Conference on 3D Vision (3DV)*. IEEE, 2021, pp. 1175–1185.

[18] A. Kendall and R. Cipolla, "Modelling uncertainty in deep learning for camera relocalization," in *2016 IEEE international conference on Robotics and Automation (ICRA)*. IEEE, 2016, pp. 4762–4769.

[19] F. Zangeneh, L. Bruns, A. Dekel, A. Pieropan, and P. Jensfelt, "A probabilistic framework for visual localization in ambiguous scenes," *arXiv preprint arXiv:2301.02086*, 2023.

[20] L. Dinh, J. Sohl-Dickstein, and S. Bengio, "Density estimation using real nvp," *arXiv preprint arXiv:1605.08803*, 2016.

[21] D. P. Kingma and P. Dhariwal, "Glow: Generative flow with invertible 1x1 convolutions," *Advances in neural information processing systems*, vol. 31, 2018.

[22] L. Ardizzone, J. Kruse, S. Wirkert, D. Rahner, E. W. Pellegrini, R. S. Klessen, L. Maier-Hein, C. Rother, and U. Köthe, "Analyzing inverse problems with invertible neural networks," *arXiv preprint arXiv:1808.04730*, 2018.

[23] L. Ardizzone, C. Lüth, J. Kruse, C. Rother, and U. Köthe, "Guided image generation with conditional invertible neural networks," *arXiv preprint arXiv:1907.02392*, 2019.

[24] C. Winkler, D. Worrall, E. Hoogeboom, and M. Welling, "Learning likelihoods with conditional normalizing flows," *arXiv preprint arXiv:1912.00042*, 2019.

[25] T. J. Adler, L. Ardizzone, A. Vemuri, L. Ayala, J. Gröhl, T. Kirchner, S. Wirkert, J. Kruse, C. Rother, U. Köthe *et al.*, "Uncertainty-aware performance assessment of optical imaging modalities with invertible neural networks," *International journal of computer assisted radiology and surgery*, vol. 14, no. 6, pp. 997–1007, 2019.

[26] T. Wehrbein, M. Rudolph, B. Rosenhahn, and B. Wandt, "Probabilistic monocular 3d human pose estimation with normalizing flows," in *International Conference on Computer Vision (ICCV)*, Oct. 2021.

[27] R. Zhao, T. Liu, J. Xiao, D. P. Lun, and K.-M. Lam, "Invertible image decolorization," *IEEE Transactions on Image Processing*, vol. 30, pp. 6081–6095, 2021.

[28] J. Arvo, "Fast random rotation matrices," in *Graphics gems III (IBM version)*. Elsevier, 1992, pp. 117–120.

[29] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin, "Attention is all you need," *Advances in neural information processing systems*, vol. 30, 2017.

[30] M. Tan and Q. Le, "Efficientnet: Rethinking model scaling for convolutional neural networks," in *International conference on machine learning*. PMLR, 2019, pp. 6105–6114.

[31] Y. Zhou, C. Barnes, J. Lu, J. Yang, and H. Li, "On the continuity of rotation representations in neural networks," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2019, pp. 5745–5753.

[32] B. Glocker, S. Izadi, J. Shotton, and A. Criminisi, "Real-time rgb-d camera relocalization," in *2013 IEEE International Symposium on Mixed and Augmented Reality (ISMAR)*. IEEE, 2013, pp. 173–179.

[33] M. O'Kelly, H. Zheng, D. Karthik, and R. Mangharam, "F1tenth: An open-source evaluation environment for continuous control and reinforcement learning," in *Proceedings of the NeurIPS 2019 Competition and Demonstration Track*, ser. Proceedings of Machine Learning Research, H. J. Escalante and R. Hadsell, Eds., vol. 123. PMLR, 08–14 Dec 2020, pp. 77–89.

[34] F. Dellaert, D. Fox, W. Burgard, and S. Thrun, "Monte carlo localization for mobile robots," in *Proceedings 1999 IEEE international conference on robotics and automation (Cat. No. 99CH36288C)*, vol. 2. IEEE, 1999, pp. 1322–1328.

[35] S. Thrun, "Probabilistic robotics," *Communications of the ACM*, vol. 45, no. 3, pp. 52–57, 2002.

[36] C. Walsh and S. Karaman, "Cddt: Fast approximate 2d ray casting for accelerated localization," vol. abs/1705.01167, 2017. [Online]. Available: http://arxiv.org/abs/1705.01167