CodeVIO: Visual-Inertial Odometry with Learned Optimizable Dense Depth

Xingxing Zuo\textsuperscript{1,2,\dagger}, Nathaniel Merrill\textsuperscript{3,\ddagger}, Wei Li\textsuperscript{4}, Yong Liu\textsuperscript{2}, Marc Pollefeys\textsuperscript{1,5}, Guoquan Huang\textsuperscript{3}

Abstract—In this work, we present a lightweight, tightly-coupled deep depth network and visual-inertial odometry (VIO) system, which can provide accurate state estimates and dense depth maps of the immediate surroundings. Leveraging the proposed lightweight Conditional Variational Autoencoder (CVAE) for depth inference and encoding, we provide the network with previously marginalized sparse features from VIO to increase the accuracy of initial depth prediction and generalization capability. The compact representation of dense depth, termed depth code, can be updated jointly with navigation states in a sliding window estimator in order to provide the dense local scene geometry. We additionally propose a novel method to obtain the CVAE’s Jacobian which is shown to be more than an order of magnitude faster than previous works, and we additionally leverage First-Estimate Jacobian (FEJ) to avoid recalculation. As opposed to previous works that rely on completely dense residuals, we propose to only provide sparse measurements to update the depth code and show through careful experimentation that our choice of sparse measurements and FEJs can still significantly improve the estimated depth maps. Our full system also exhibits state-of-the-art pose estimation accuracy, and we show that it can run in real-time with single-thread execution while utilizing GPU acceleration only for the network and code Jacobian.

I. INTRODUCTION

Accurate pose estimation and dense depth estimation are essential to a wide range of robotic applications such as obstacle avoidance and path planning. Recently, utilizing Conditional Variational Autoencoder (CVAE) networks as a differentiable manifold to compress dense depth maps into a far lower-dimensional subspace, has shown promising results \cite{1} and even shown real-time performance when leveraging a desktop graphics card and incremental smoothing \cite{2}. However, these methods only utilize camera information, and rely solely on the depth network to recover scene information from learned priors. Additionally, even the real-time method \cite{2} still relies heavily on GPU acceleration to perform dense warping to extract whole-image photometric residuals, and to our knowledge, no work thus far has presented a method to efficiently calculate the network Jacobian.

To address some of these shortcomings, we take a completely different approach, and leverage a lightweight network and IMU-driven EKF-based estimator to gain some easy speedups and provide a metric scale to the whole system. Furthermore, we propose a new method to recover the approximate VAE decoder Jacobian with near-perfect accuracy and over an order of magnitude speedup over the traditional chainrule method provided by deep learning libraries, and show that only providing sparse measurements to the VAE can still improve performance considerably and in an efficient manner. Our main contributions are:

- We develop an efficient VIO system that jointly estimates a scaled local dense reconstruction along with sparse features and IMU poses in a tightly-coupled manner over a sliding window in real time using single thread CPU execution and only utilizing the GPU to accelerate the network inference and VAE Jacobian.
- A lightweight CVAE model is proposed, which leverages depthwise-separable convolutions and additive skip connections to enable highly-efficient inference while retaining a high level of accuracy. Through CVAE, a dense depth map is encoded into a low-dimensional depth code, which can be jointly updated with navigation states.
- We present a new method to recover the approximate VAE decoder Jacobian, which has near-perfect accuracy and achieves over an order of magnitude speedup over the chainrule method previously adopted. Also, due to our use of First-Estimate Jacobians (FEJ), the Jacobian...
Due to our fusion of IMU information, we propose to feed marginalized sparse features from recent frames along with the images to the network to increase the accuracy of the initial depth map and thus the zero code prior. Furthermore, we only use sparse updates on the codes that are estimated in order to further reduce the required computation and necessity for GPU acceleration.

II. RELATED WORK

Mapping with the aid of IMU is appealing. A dense mapping method deployed on aerial robots is proposed in [3]. Poses are estimated in a factor graph optimization-based VIO. Then the scale determined poses from VIO are passed into a motion stereo based depth estimation module. In this way, the scale information from IMU is just implicitly incorporated into the dense depth through motion stereo. Kimera [4] is another VI-SLAM system with the capability of mesh reconstruction, which supports both monocular and stereo camera. Along with pose estimation, a 3D mesh module for dense reconstruction is enabled by sparse points aided 2D Delaunay triangulation.

Without additional sensors, neural networks can also recover monocular depth with pseudo scale, which is learned in a data-driven fashion, such as [5, 6]. In FastDepth [6], an efficient and lightweight encoder-decoder network architecture for depth prediction is proposed. It is fast and applicable to embedded systems for real-time depth prediction. However, the accuracy and reliability of monocular depth prediction cannot be guaranteed in practical application. To address this issue, some works resort to scale determined sparse depth. For example, Ma and Karaman [7] propose a deep CNN for dense depth prediction from monocular image and sparse depth, while Tang et al. [8] propose a learning guided convolutional network for depth completion.

Due to the significant progress in learning-based depth prediction, SLAM with learned dense depth are also explored in many existing works [9–12]. CNN-SLAM [9] leverages single image depth prediction and perform dense alignment for a dense 3D reconstruction in a full SLAM system. In [10], deep depth predictions are incorporated into a directed method based monocular visual optometry as direct virtual stereo measurements for dense mapping. However, in all the method mentioned above, the dense depth are computationally intensive to process due to its high dimensionality. To address this issue, a milestone method, CodeSLAM [1], is proposed. CodeSLAM introduces the concept of learning a compact optimizable representation of dense depth through conditional variational auto-encoder (CVAE). Dense depth is encoded in a compact code and can be dexterously incorporated for solving a dense structure-from-motion problem. CodeSLAM is computationally intensive, and thus its follow-up work DeepFactors [2] is developed with a capability of real-time SLAM in a factor graph-based optimization framework. However, a big portion of computation in the system, including image warping and full-image tracking, is conducted on the GPU, and the network Jacobian calculation is still a bottleneck. CodeSLAM and DeepFactors are the two closest work to our CodeVIO. However, both can only perform up-to-scale dense reconstruction, since only RGB images are utilized in the inference stage. In contrast, our proposed CodeVIO offers metric dense depth and shows great generalization capability.

III. FAST DEPTH INFERENCE AND ENCODING

Inspired by CodeSLAM [1], the proposed network architecture contains two streams of information (see Fig. 2): (i) a modified FastDepth [6] with the pruned architecture, where the input of RGB or grayscale image concatenated with a sparse depth map from VIO is used to obtain deep feature map for our second stream as well as estimate a depth inference uncertainty map; and (ii) a variational auto-encoder (VAE), which is conditioned on the learned deep features from the first stream, and is trained to encode dense depth maps. During training and validation, we sample sparse depth points from the groundtruth dense depth maps to allow training on a general RGBD dataset. To imitate the sparse depths from VIO, for training, we detect FAST corner features [13] from the images and randomly sample 50–200 of them. To mitigate the sensitivity to the noisy depth from VIO, we inject Gaussian noises with a standard deviation of 0.1m to the sampled sparse depth in the training phase. This sparse depth sampling strategy mostly avoids overfitting to idealistic uniformly-distributed true depth values with fixed sparsity as in existing literature [7].

To ensure low latency, all the convolutional layers are depthwise separable [14], instead of standard convolutional layers. An efficient MobileNet [14] with depthwise decomposition serves as the encoders in both the UNet feature extraction stream and the auto-encoder stream. The upsample layers consist of depthwise separable convolutions with a kernel size of 5 and nearest neighbour based interpolation. The output of the upsample layers has decreased channels compared to the input channels. In the VAE stream, dense depth is encoded into a low dimensional depth code at the bottleneck of VAE in the training phase. Then dense depth can be predicted from the zero code
conditioned on the deep feature maps from the UNet. In initial experimentation, we toyed with leveraging the linear decoder idea of [1], however, due to the proposed novel method to efficiently obtain the Jacobian as well as our use of FEJ [15–17], we actually do not explicitly use any property of the linear decoder, and simply leave it in our network as it did not significantly hurt the accuracy.

To lower computations, all the skip connections are via addition rather than concatenation as in CodeSLAM [1]. The skip connections provide low-level gradient and scale information from the UNet encoder for its decoder to preserve details from the input image and sparse depth. Similar to [1], to preserve the influence of UNet feature maps on the VAE’s code Jacobian, we additionally add the element-wise multiplication term to the additive skip connection, thus compressing the concatenated skip connections of CodeSLAM by a factor of three while preserving most of the information. Similar to [6], we found that the additive skip connections speed up the network significantly with little effect on accuracy. We also found that two skip connections speed up the network significantly with little information. Similar to [6], we found that the additive skip connection, thus compressing the concatenated skip connections of CodeSLAM by a factor of three while preserving most of the information. Similar to [6], we found that the additive skip connections speed up the network significantly with little effect on accuracy. We also found that two skip connections speed up the network significantly with little information.

The training loss in our network is the summation of the loss in the depth stream and the loss in the VAE. The training loss is averaged for each batch during training, and the loss is averaged for each batch during training, and the depth values are thresholded accordingly to prevent an infinity log. To stabilize training, we leverage a KL annealing method to efficiently obtain the Jacobian as well as our use of FEJ [15–17], we actually do not explicitly use any property of the linear decoder, and simply leave it in our network as it did not significantly hurt the accuracy.

\[ L_{rec} = \frac{1}{|\Omega|} \sum_{x \in \Omega} \left[ \log(D(x)) - \log(\hat{D}(x)) \right] + \hat{B}_\text{log}(x) \]

where \( x \in \Omega \) denotes every pixel in the output feature maps. The loss is averaged for each batch during training, and the depth values are thresholded accordingly to prevent an infinity log. To stabilize training, we leverage a KL annealing method to efficiently obtain the Jacobian as well as our use of FEJ [15–17], we actually do not explicitly use any property of the linear decoder, and simply leave it in our network as it did not significantly hurt the accuracy.

The training loss in our network is the summation of the depth recovery loss and the VAE’s KL divergence loss [18]. The KL divergence loss closes the gap of the depth code distribution to standard normal distribution. We predict the logarithm of the inverse depth values (\( D_{\text{loginv}} \)) from the VAE rather than the raw depth values or the proximity proposed in [1], and the UNet predicts the logarithm of depth uncertainty value (\( \hat{B}_{\text{log}} \)). The depth recovery loss is given by:

\[ x_f = \begin{bmatrix} 4p_0^T & 4p_1^T & \cdots & 4p_{f-1}^T \end{bmatrix}^T \]

\[ x_{\text{code}} = \begin{bmatrix} c_k^T & c_{k-\ell}^T & \cdots & c_{k-np}^T \end{bmatrix}^T \]

In the above expressions, besides the standard VIO notations that can be found in [21], we here focus on the new state, i.e., CVAE code \( x_{\text{code}} \). The depth codes are extremely low-dimensional compared to the depth maps they encode (32 vs 224^2 = 50176 in our case) and can be efficiently updated with sparse measurements in the sliding-window, which is considerably different than the dense measurements used by similar works. While the codes are low dimensional, including one for every image still adds considerable computation to the EKF update, and is highly redundant – as the depth maps will overlap more than necessary. To this end, we use a constant interval keyframing strategy in order to simplify the logic of ensuring a code is always in the sliding window.

### A. Optimizable Dense Depth and Inertial Navigation

We perform state estimation and dense local mapping in the MSCKF framework, leveraging FEJ to improve consistency as well as negate the need to ever re-calculate a code’s Jacobian. The IMU propagation phase is the standard forward integration in MSCKF. The IMU states \( x_I \) and their corresponding covariances are propagated from the last image time instant in IMU time axis to the current time. Then, stochastic cloning of the propagated IMU pose is performed for state augmentation. If the image frame is selected as a keyframe, we will also augment the state vector with the depth code of this image.

The depth code is initialized to be a zero vector and assigned a covariance of \( \sigma_c^2 I \), which coincides with the assumption that the depth code in the VAE is under the standard normal distribution. In practice, we use \( \sigma_c > 1 \) to account for the fact that the VAE does not perfectly follow the standard normal distribution. All the states, including the depth codes, are updated with measurements in the update phase.

By encoding the dense depth of keyframes with the low dimensional depth code via VAE, the dense depth is a function of the raw image \( M \), sparse depth \( S \), and the depth code estimate \( \hat{c} \). Thus we write

\[ \hat{D} = f(\hat{c}) = f(M, S, \hat{c}) \]

in order to simplify notation. Here \( f \) is simply the VAE decoder function. The density map is a nonlinear function of the raw image \( M \) and sparse depth \( S \) due to nonlinear activations in the UNet. It is also important to note that, although a linear decoder is employed in the VAE, the final operation to obtain the network output in depth form

\[ D[u, v] = \exp(-D_{\text{loginv}}[u, v]) \]

makes the depth map a nonlinear function of the code vector as well.

#### 1) Reprojection Error and Sparse Geometric Update

For a triangulated 3D sparse visual feature \( A\mathbf{p}_j \) observed in image \( \mathbf{M}_j \) (at camera frame \( \{C_j\} \) ) with observation \( \mathbf{z}_{\text{rep},f_j} = \big[u'_i, v'_i\big]_\text{T} \), the reprojection residual is [23]:

\[ r_{i,j}^{\text{rep}} = z_{\text{rep},f_j} - \hat{h}(\mathbf{x}_j, A\mathbf{p}_j) \approx H_{x_j} \hat{x}_j + \mathbf{H}_{x_j} A\mathbf{p}_j + \mathbf{n}_r \]

where \( \mathbf{x}_j = \begin{bmatrix} \mathbf{q}_j^T & \mathbf{G}_{P_{k-1}}^T \cdots \mathbf{G}_{P_{k-np}}^T \end{bmatrix}^T \) includes the cloned pose corresponding to \( \{C_j\} \) as well as the necessary the
camera intrinsics $\zeta$ and the IMU-to-camera extrinsic parameters to project $A^p_{f_i}$ onto the image plane. Additionally, if the anchor frame $\{A\}$ is not the static global frame, then $x_j$ also contains the camera clone pose corresponding to $\{A\}$. $H_{x_j}$ and $H_f$ are the Jacobians of the reprojection error residual with respect to $x_j$ and $A^p_{f_j}$. $u_r$ denotes the feature tracking noise under a Gaussian distribution with isotropic covariance matrix $R = \sigma^2_{im}I_2$.

If $\{C_j\}$ is also a keyframe with a corresponding depth map $C_j^D$, we can formulate a sparse geometric constraint for the observed visual feature $A^p_{f_j}$, by now considering $\mathbf{z}_{rep,f_j}$ as the noise-free $[u, v]^T$ (dropping the subscripts to simplify notation), and utilizing the optimization of the depth value $c^i_jD[u, v]$ at the nearest pixel location of the depth map, which of course noisy due to the network weights, we can write the sparse geometric residual as

$$r^geo_{i,j,a,b} = C_j^D[u, v] - g(\hat{x}_j, A^p_{f_j}) = C_j^D[u, v] - e_3^T \left( C_j^A R A^p_{f_j} + C_j^A C_j^A \right) + n_{d_i}^j \approx G_{e_j} \hat{c}_j + G_{x_j} \hat{x}_j + G_{f_j} A^p_{f_j} + n_{d_i}^j$$

where $e_3 = [0 0 1]^T$, and $C_j^A, C_j^A C_j^A$ are functions of $x_j$. Moreover, $G_{e_j}, G_{x_j}$, and $G_{f_j}$ are the Jacobians of depth geometry measurement with respect to the error states of $c_j, x_j,$ and $A^p_{f_j}$, respectively. In this case, $G_{e_j}$ is a single row of the Jacobian of predicted depth with respect to its code as described in Sec. IV-B, which is evaluated at the corresponding row for $[u, v]^T$. The noise of the depth geometry measurement is modeled by $n_{d_i}^j \sim (0, \sigma_{d_i}^2)$, where $\sigma_{d_i} \approx \exp(c_j \delta_{log}(u_i, v_i))$ and thus the measurement covariance can be directly fetched from the predicted depth uncertainty. In contrast to reprojection noise, the depth noise generally varies for different observations of the same feature (i.e. $\sigma_{d_i}^2 \neq \sigma_{d_j}^2$ for $a \neq b$). Since this geometric update is performed mainly with the MSCKF features, it should be performed in the same EKF update as the reprojection error update – as the MSCKF features will be marginalized directly afterwards. Furthermore, any feature track location $[u, v]^T$ used in this update is ignored later on to prevent reusing information. In practice, the depth measurements across images are correlated at least to some degree due to sharing the same network weights; however, this can be remedied by inflating the noise.

Note that the geometric observations of depth map $C_j^D = f(M_{i_j}, S_j, \hat{c}_j)$ are related with the sparse depth map $S_j$, which is generated by projecting previously marginalized MSCKF features onto image plane, and thus never includes $A^p_{f_j}$. Therefore, we consider the image and sparse depth map as constant prior information in our system. Specifically, they only serve to increase the accuracy of the zero code depth map estimate and subsequently the quality of the zero code as initial guess for $c_i$, which need to be good in order for the EKF update with FEJ Jacobians to work properly.

2) Depth Consistency Update: The depth values of the tracked features should be consistent among different keyframes. To jointly improve the quality of recovered dense depths, we formulate a depth consistency measurement for all tracked feature locations that were not already used in the sparse geometric update. For a pair of feature tracks observed as $o_i = [u_i, v_i]^T, o_j = [u_j, v_j]^T$ in the image planes of camera frames $\{C_a\}$ and $\{C_b\}$, the depth consistency residual becomes

$$r^cons_{i,j,a,b} = C_a D_j - e_i^T w(\hat{x}_{ab}, C_a D_i, o_i)$$

$$= C_a D_j - e_j^T w(\hat{x}_{ab}, C_b D_i + n_i, o_i) + n_j$$

$$\approx W_{cb} \hat{c}_b + W_{uxab} \hat{x}_{ab} + W_{wc} \hat{c}_a + n_{i,j}$$

where $C_a D_j [u_i, v_i]$ is denoted by $C_a D_i$ for clarity, and similarly for $b$ and $j$. Additionally, $w$ projects the image coordinate $o_i$ and depth $C_a D_i$ into the $\{C_a\}$ frame, and then into the $\{C_b\}$ frame, and thus is a function of calibration states, as well as the camera clones corresponding to $\{C_a\}$ and $\{C_b\}$, which are all encapsulated by in $\hat{x}_{ab}$ here. The Jacobians $W_{cb}, W_{uxab}, W_{wc}$ are, as before, the measurement with respect to each of the subscripts’ error states. Again, the Jacobians for the codes require a single row of the Jacobian of the VAE decoder with respect to the code. Similar to the sparse geometric update the variances from the noises $n_i, n_j$ are taken from the UNet’s estimated uncertainty map, and the variance for the compounded noise $n_{i,j}$ is calculated with a simple covariance propagation.

B. Efficient Network Jacobian Computation

It is important to note that all of the previously mentioned measurement updates involving the depth codes require the Jacobian $\frac{\partial D}{\partial c_i}$ of the predicted depth map with respect to the code vector error state – evaluated at the current code estimate. This is definitely a difficult task to accomplish in real time, as deep learning libraries are optimized to retrieve gradient vectors, but not full Jacobian matrices. Czarnecki et al. [2] report that the Jacobian computation in TensorFow takes over 300ms on a GTX 1080Ti GPU, which is much too slow for our needs.

Since we only require the Jacobian of the decoder with respect to the input depth code – and not all the intermediate weights – we actually do not need to calculate a full chain rule. Instead, we propose to simply estimate the Jacobian using the following finite difference equation.

$$\frac{\partial D}{\partial c_i} \approx f(\hat{c} + \delta_{c_i}) - \hat{D}$$

for some typically small $\delta$, and where $e_i$ is equivalent to the $i$th column of the $c \times c$ identity matrix $I$. This equation represents the derivative of the depthmap for a single element $c_i$ of the code $c \in \mathbb{R}^c$. Since the depth decoder function $f$ is a neural network, which is designed to process batch data, it seems perfectly suited for this application – which only requires running a mini batch of size $c$ ($c = 32$ in our case) for just the lightweight VAE decoder, as the UNet skip connections are reused from predicting the initial zero-code depth estimate. The proposed method can estimate the full dense Jacobian with near-perfect precision (as shown later in the experiments) while only requiring a single batched forward pass through the network. We found that this operation only takes roughly 10ms on a desktop-grade GPU (i.e., GTX 1080Ti or RTX 2070 super) and 40ms on the GTX 1060 on one of our laptops. This is over an order of magnitude speedup from using the chain rule Jacobians calculated by a network backward pass.
TABLE I: Performance of depth prediction and encoding methods on the NYU Depth V2 dataset. RMSE and iRMSE are in meters, and \( \delta \) in \( \text{m}^{-1} \), respectively. GPU and CPU timings are in milliseconds.

<table>
<thead>
<tr>
<th>Methods</th>
<th>RMSE ( \delta )</th>
<th>iRMSE ( \delta )</th>
<th>MAPE ( \delta )</th>
<th>( \delta )</th>
<th>CPU</th>
<th>GPU</th>
</tr>
</thead>
<tbody>
<tr>
<td>FastDepth-RGB</td>
<td>0.456</td>
<td>0.080</td>
<td>0.115</td>
<td>0.873</td>
<td>0.968</td>
<td>0.991</td>
</tr>
<tr>
<td>Ours-RGB</td>
<td>0.545</td>
<td>0.080</td>
<td>0.137</td>
<td>0.870</td>
<td>0.951</td>
<td>0.985</td>
</tr>
<tr>
<td>Ours-RGB-Sp</td>
<td>0.316</td>
<td>0.052</td>
<td>0.066</td>
<td>0.944</td>
<td>0.986</td>
<td>0.996</td>
</tr>
<tr>
<td>Ours-Gray</td>
<td>0.535</td>
<td>0.089</td>
<td>0.133</td>
<td>0.833</td>
<td>0.953</td>
<td>0.985</td>
</tr>
<tr>
<td>Ours-Gray-Sp</td>
<td>0.315</td>
<td>0.055</td>
<td>0.071</td>
<td>0.938</td>
<td>0.985</td>
<td>0.995</td>
</tr>
</tbody>
</table>

V. EXPERIMENTAL RESULTS

A. Evaluation of Depth Prediction

We extensively evaluate the network used in this work, regarding the depth prediction accuracy and run time. All the evaluations in this section are conducted on a commercial desktop with a GTX 1080Ti GPU and Intel i7-8086k CPU@4.0GHz. Some standard metrics from the literature are used to evaluate the accuracy of depth prediction, including root mean squared error of depth (RMSE), root mean squared error of inverse depth (iRMSE), mean absolute percentage error (MAPE), and \( \delta \), which is the percentage of predicted depth with a certain relative error [24]. To evaluate the computational intensity, the run time evaluated on different hardwares are shown, including the 1080Ti GPU, only CPU with 4 threads, and only CPU with one single thread. The results are shown in Table I. Five networks are compared including FastDepth [6] with input of RGB image only (FastDepth-RGB), the proposed depth-wise convolution based CVAE with RGB input only (Ours-RGB), the proposed CVAE with both RGB and sparse depth (Ours-RGB-Sp), the proposed CVAE with grayscale image only (Ours-Gray), and the proposed CVAE with both grayscale image and sparse depth (Ours-Gray-Sp).

TABLE II: Performance of the depth estimation on EuRoC dataset. Entries for “zero code”/“updated” prefixed with “Sp” have grayscale and sparse depth input, while those without have grayscale input. For each sequence, the method with the best performance is marked in green with the best score in blue. RMSE and iRMSE are in meters and \( \text{m}^{-1} \), respectively.

<table>
<thead>
<tr>
<th>Methods</th>
<th>RMSE ( \delta )</th>
<th>iRMSE ( \delta )</th>
<th>MAPE ( \delta )</th>
<th>( \delta )</th>
<th>CPU</th>
<th>GPU</th>
</tr>
</thead>
<tbody>
<tr>
<td>V1-01</td>
<td>0.2402</td>
<td>0.0499</td>
<td>0.0457</td>
<td>0.9666</td>
<td>0.9791</td>
<td>0.9847</td>
</tr>
<tr>
<td>Update</td>
<td>0.8168</td>
<td>0.1706</td>
<td>0.2464</td>
<td>0.5222</td>
<td>0.8002</td>
<td>0.9295</td>
</tr>
<tr>
<td>V1-03</td>
<td>0.8001</td>
<td>0.1692</td>
<td>0.2413</td>
<td>0.5304</td>
<td>0.8073</td>
<td>0.9338</td>
</tr>
<tr>
<td>Update</td>
<td>0.4867</td>
<td>0.0932</td>
<td>0.1108</td>
<td>0.8644</td>
<td>0.9499</td>
<td>0.9771</td>
</tr>
<tr>
<td>V1-02</td>
<td>0.3991</td>
<td>0.0783</td>
<td>0.0923</td>
<td>0.8990</td>
<td>0.9428</td>
<td>0.9688</td>
</tr>
<tr>
<td>Update</td>
<td>0.8037</td>
<td>0.1584</td>
<td>0.2417</td>
<td>0.5122</td>
<td>0.8153</td>
<td>0.9443</td>
</tr>
<tr>
<td>V2-01</td>
<td>0.7923</td>
<td>0.1573</td>
<td>0.2386</td>
<td>0.5192</td>
<td>0.8200</td>
<td>0.9460</td>
</tr>
<tr>
<td>Update</td>
<td>0.7212</td>
<td>0.1245</td>
<td>0.1992</td>
<td>0.7599</td>
<td>0.8944</td>
<td>0.9500</td>
</tr>
<tr>
<td>Update</td>
<td>0.6018</td>
<td>0.1177</td>
<td>0.1697</td>
<td>0.7965</td>
<td>0.9185</td>
<td>0.9643</td>
</tr>
</tbody>
</table>

Typically, in finite difference equations Eq. (12), a very small \( \delta \) must be used to perturb the inputs in order to obtain reasonable accuracy (not to be confused with the \( \delta \) accuracy metric in the other tables). However, in our case, we found that a small \( \delta \) actually made the Jacobian less accurate than the chain rule Jacobian taken from PyTorch autograd. In order to find the best \( \delta \) value, we use PyTorch autograd in batch mode to calculate the first 32 rows of the Jacobian using the batch input we used for the finite difference Jacobian (clearly before applying the perturbation). We only do the first 32 rows so that we can reuse the same batch as our finite difference method, as calculating the full Jacobian with PyTorch would require many more forward passes. We test with the first 100 images in the EuRoC V2_01 dataset, and compare the max, min, and average element-wise absolute error between the autograd Jacobian and the first 32 rows of ours over a range of different \( \delta \) values.
the CV AE decoder maps a 32-dimensional vector to $224$ in the input code does not change the output by much, since we suspect that this is due to the fact that a small perturbation to the zero code predictions is for each of the V1 sequences, and the V2 is on the bottom.

Fig. 5 summarizes our findings, where it can be seen that the accuracy of the finite difference Jacobian is best with $\delta$ between roughly 0.5 and 1.0, which is many orders of magnitude higher than we initially expected from observing other implementations of finite difference Jacobians. We suspect that this is due to the fact that a small perturbation in the input code does not change the output by much, since the CVAE decoder maps a 32-dimensional vector to $224^2$ outputs, so it must compress information about large groups of pixels into each single code entry. Due to this, we chose $\delta = 0.666$ in our experiments, as it falls in the most accurate range of $\delta$ values.

C. Full Evaluation on EuRoC Dataset

In order to gauge the performance of the whole system, we test it on all the Vicon Room sequences (see Fig. 1) of the EuRoC dataset [26]. These sequences include an accurate LiDAR point cloud map, which we used to obtain the ground truth depths. While some works have focused on the reconstruction accuracy compared to the whole LiDAR point cloud map on this dataset [27], we instead are only concerned with the accuracy of the depth maps – as we do not perform any global mapping. To this end, we used the ground truth camera poses to project the point cloud into the undistorted image plane – keeping the minimum depth per location in order to account for occlusions. Note that the feature tracking, IMU propagation, temporary MSCKF feature update with sparse geometry, “long-lived” SLAM feature update, zero code initialization (network inference and Jacobian calculation), depth consistency update, and the re-inference of dense depth maps after their codes are updated. The mean run time for the entire loop is $0.0447 \pm 0.0226$ seconds, which is suitable for applications requiring real-time capability. It should be noted that our current implementation is not highly-optimized and can still be further improved.

VI. CONCLUSIONS AND FUTURE WORK

In this paper, we have designed a real-time VIO system with the capability of dense local mapping. An efficient CVAE neural network for depth prediction and encoding is proposed, which also takes advantage of the sparse depth from VIO and generalize well on a completely unseen dataset. Dense depth is compactly encoded in a low dimensional code, which is optimizable and estimated with inertial navigation states in a tightly-coupled lightweight MSCKF framework. In addition, a novel and efficient network Jacobian computation method is proposed, which is over an order of magnitude speedup over the chainrule method. This work targets local dense mapping, and we leave full SLAM with a global dense depth map as future work.