Robust Monocular Visual-Inertial Depth Completion for Embedded Systems

Nathaniel Merrill*, Patrick Geneva*, and Guoquan Huang

***Abstract*— In this work we augment our prior state-of-the-art visual-inertial odometry (VIO) system, OpenVINS [1], to produce accurate dense depth by filling in sparse depth estimates (depth completion) from VIO with image guidance – all while focusing on enabling real-time performance of the full VIO+depth system on embedded devices. We show that noisy depth values with varying sparsity produced from a VIO system can not only hurt the accuracy of predicted dense depth maps, but also make them considerably worse than those from an image-only depth network with the same underlying architecture. We investigate this sensitivity on both an outdoor simulated and indoor handheld RGB-D dataset, and present simple yet effective solutions to address these shortcomings of depth completion networks. The key changes to our state-of-the-art VIO system required to provide high quality sparse depths for the network while still enabling efficient state estimation on embedded devices are discussed. A comprehensive computational analysis is performed over different embedded devices to demonstrate the efficiency and accuracy of the proposed VIO depth completion system.**

I. Introduction

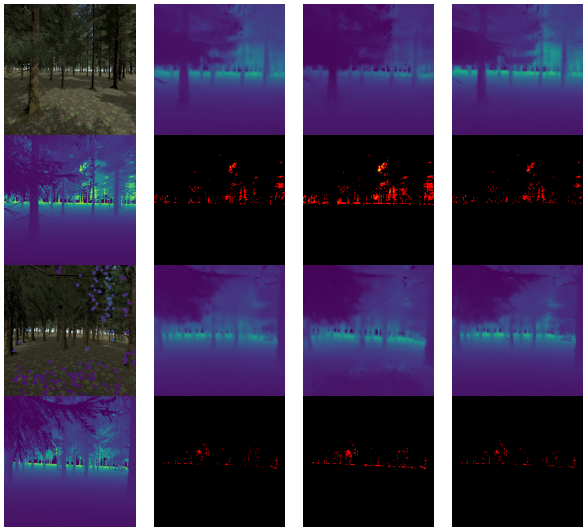
Traditionally, real-time dense depth is achieved through depth sensors, posing hardware constraints which may not amenable for embedded resource constrained systems such as micro-aerial vehicles (MAVs) and mobile augmented-reality (AR) devices. Recent works have leveraged deep learning to predict depth from single images [2]–[6], image sequences [7]–[9], and also sparse depth maps paired optionally with images (also known as depth completion) [10]–[15]. A select few of these methods [6], [13] are particularly appealing as lightweight network architectures are used. In this work, we augment our prior state-of-the-art VIO system, OpenVINS, with the ability to produce real-time dense depth, while focusing on enabling real-time state estimation and dense depth completion on embedded systems.

In particular, dense depth completion from images combined with sparse depths generated from VIO systems has shown promising accuracy gains over pure image-based depth prediction networks [13]–[15]. In this work we extend the FastDepth [6] architecture to support sparse depth inputs along with color images and investigate sensitivities of this network to VIO errors on resource-constrained devices. The main contributions of this work include:

- We show that noisy depth values with varying sparsity produced by VIO can not only hurt the accuracy of the completed depth maps, but also make them considerably worse than the predictions from an image-only depth network with the same underlying architecture.

*These authors contributed equally to this work.

The authors are with the Robot Perception and Navigation Group (RPNG), University of Delaware, Newark, DE 19716, USA. {nmerrill,pgeneva,ghuang}@udel.edu



(a) (b) (c) (d)

Fig. 1: **(a)** The example RGBsD inputs from VIO along with the groundtruth depth maps. **(b)** The prediction from the RGB only network with a heat map of the per-pixel absolute error (white is larger). **(c)** The prediction from the depth completion network which has been trained without noise performs worse than the RGB network. **(d)** The proposed network trained to be robust to extreme varying levels of sparsity as well as the noisy depth values.

- We propose novel solutions to address these shortcomings and robustify the network, which are generalizable to nearly any sparse depth completion network and dataset without altering the network architecture. The proposed methods are presented and evaluated on both an outdoor MAV simulated and indoor handheld RGBD dataset, with both simulated and real VIO depth inputs.
- We introduce some key changes to our OpenVINS [1], required to provide high-quality sparse depths inputs for the network and enable efficient VIO depth completion on embedded devices (<10-15W). We show through comprehensive timing experiments that the proposed VIO depth completion system can achieve this goal.

II. Related Works

Pairing images and sparse depth as input to depth estimation networks has been of particular interest recently due to its promise of accurate dense depth map predictions – typically with minimal additional computation compared to just image inputs. While many works have studied depth from a single image [2]–[6], [16], [17], the inclusion of a sparse depth source (e.g. VIO, selective stereo matching, range sensor) [10]–[12], [18]–[21] has shown far better results due to the fact that the network has access to some known depth values – thus relaxing the ill-poised problem

of single-view depth estimation. Ma and Karaman [10], [11] introduced a sparse-to-dense ResNet-based model which combined a 4-channel input of RGB and sparse depth points and demonstrated that coupling the two as a single input in an end-to-end fashion outperforms the state-of-the-art and reduces prediction error compared to their RGB-only counterparts. Note that while they reported that training with extreme sparsity performs worse than the RGB counterpart, this is quite different from our experiments, which shows this phenomena without even varying the sparsity.

Significant research efforts have investigated sparse depth sensitivities [22]–[28]. Jaritz et al. [22] showed that changing the training procedure to vary the level of sparse point densities greatly improves robustness to varying levels of sparse points during test time. Cheng et al. [23] investigated the sensitivity of LiDAR-stereo depth network completion algorithms to noisy LiDAR points and sensor extrinsic misalignment and addressed sensor misalignment through a training strategy which gradually removed inconsistent noisy LiDAR points. In this work, we investigate the sensitivity of a lightweight model architecture to the noises present in VIO depths with the goal of running the full system on an embedded platform.

The coupling of visual-inertial state estimation and dense depth prediction has seen recent interest. A popular technique is to leverage meshing of sparse points [14], [29]. A major drawback is that the resolution of the mesh is limited to the density of sparse points, which limits its applicability to outdoor non-planar environments. Sartipi et al. [15] looked at directly leveraging the sparse points from VIO through a multi-stage system which detects planes, surface normals, and finally a dense depthmap. However, this method has high computational complexity and only focuses on indoor planar environments.

As for deep depth estimation on embedded devices, Wofk et al. [6] introduced the FastDepth architecture which focused on enabling real-time depth inference on embedded platforms. They successfully reduced the computational complexity of a dense RGB depth network to enable real-time inference on an NVIDIA Jetson TX2 GPU. Teixeira et al. [13] presented a lightweight network architecture that could perform image-guided depth completion from VIO depths at 15Hz on a TX2; however, they suffered from significant accuracy loss when going from the uniformly sampled depths used in training to the real VIO depths. Due to its superior speed, in this paper, we utilize the FastDepth architecture to support the inclusion of sparse depth from a VIO system with the goal of reducing predicted depth errors.

III. Sparse Depth Completion

To enable real-time dense depth completion on embedded platforms we leverage the final pruned FastDepth network architecture. In this section, we identify and investigate the particular problems that arise from attempting to use sparse OpenVINS VIO depth values as input to the depth completion network, and show that naïve image-guided depth completion may become worse than RGB-only depth.

We investigate two methods of which to modify the hourglass FastDepth architecture to allow for combined image and sparse depth input. The first is the original method of Ma et al. [10] which involves simply concatenating the sparse depth map with the input image and inputting this four channel image. The second is the method of late fusion, similar to Jaritz et al. [22], which has two separate encoders for the image and sparse depth. Unlike [22], which concatenates the skip connections from the two encoders to be fed into the decoder, we use element-wise addition to speed up the computation (as in [6]).

A. OpenVINS Modifications

At its core OpenVINS [1] is an on-manifold modular EKF-based sliding-window visual-inertial estimator. There are two different types of features used: 1) SLAM features are inserted into the state vector and updated with new measurements in future timesteps directly, and 2) MSCKF features are short-term or extra features whose measurements are batch processed using the MSCKF [30] null-space projection to remove the need to include them in the state vector. The former become computationally cheaper if they are created for features which have long feature tracks, while the latter are cheaper when there are large numbers of short feature tracks [31]. Thus a combination of these two feature types is typically used to maximize performance.

To facilitate the proposed depth completion network and reduce computation, a few key changes are made. First, we introduce a limit to the number of MSCKF features which can be used in a single update step, with features being selected based on maximum track length. Leftover "actively tracked" features will be postponed to a future update. Second, we perform the process of re-triangulation after each feature update over *all* actively tracked features which can be triangulated. After triangulation, features are projected into the current camera frame using the currently estimated poses and calibration values, and then sent to the network in the form of a sparse depth image. We found that this re-triangulation typically only takes 1-2ms for 500 features, which is computationally cheap and allows for downstream applications (e.g., depth completion, loop-closure detection) to leverage the pointcloud. In practice this re-triangulation can be performed before or after state update to either enable lower latency for depth prediction or more accurate sparse depths (which we use as default).

B. Simulation Analysis

Our analysis is performed on a simulated outdoor forest environment, as compared to existing RGBD datasets, to provide a challenging depth estimation scenario with noise-free and fully dense ground truth depth maps. Shown in Figure 1 (left column), this environment presents a particular challenge due to the density of the forest and range of depths observed (maximum depth of 65 meters) – as well as the large roll and pitch variations that we allow. There is a high density of small branches and leaves which presents a challenge in recovering their depths, but also are crucial to recover accurately in cases where the predicted depth map is used for obstacle avoidance. We collect 29,909 images in total by randomly placing the camera in the forest world, with random position and orientations of roll $\pm 25^\circ$ and pitch $\pm 45^\circ$, and excluding any images where the camera landed

inside of an object. 19,716 and 1195 images are respectively used for training and validation, and 8,998 are reserved for testing. We use a relatively large testing split in order to ensure that none of the results are anomalies from overfitting. Due to the challenging environment with clutter from the trees, large depth ranges, and large amounts of roll and pitch variations, the networks struggle to achieve metrics on the order of public benchmarks that are easier due to their limited orientation changes and relatively small clutter [19].

1) Training

We train all the networks in this paper with PyTorch [32], using the Adam optimizer [33] with the default parameters and a batch size of 64. Unless otherwise specified, all of the networks discussed are initialized with weights from a MobileNet model [34] pretrained on the ImageNet [35] and trained for 30 epochs with the ℓ_1 loss and the data augmentations as in [2]. As opposed to the original training method of FastDepth, which involves training the model with the full MobileNet model, pruning, and then fine tuning, we simply use the pruned FastDepth model architecture, and copy over the first c_p channels from any layer of the MobileNet model that has c layers, where $c_p \leq c$. This helps to streamline the training process and achieves the same accuracy as the original work. Note that in both training and testing, when noises are present, the same random seed is used to provide a fair relative comparison.

2) Sensitivity to Sampling Methods

We first investigate the sensitivity of the depth completion networks to sampling methods alone by fixing the number of points and using the true depth values. We train both the baseline RGB FastDepth and the RGB-sparse depth (RGBsD) networks with both the shared encoder and late fusion architectures, using some standard metrics used in the depth estimation literature[1]. The following training depth sampling strategies are investigated: 1) points sampled uniformly according to the original method of Ma et al. [10] (unif), 2) raw FAST corner locations (fast), and 3) FAST corners with injected noise to model VIO depths (robust), which is explained in more detail in the next section. 500 samples are used in each case, with slight variations in the uniform samples as in [10], as well as in the FAST corners due to a lack of gradients in darker areas. We test these models with each one of these sampling strategies in order to observe how the sampling affects each one individually. It is clear from Table I that just sending FAST corner features to the network trained with uniform samples quickly degrades the performance past that of the RGB network. While it has been shown in [13], [15] that sending corner features with and without noise to a network trained with uniform data can hurt the performance, to our knowledge this is the *first* time it has been observed to perform significantly worse than a comparable RGB network. Since we want to leverage sparse VIO depth of FAST features, we could use the system trained on the true FAST corners, but this ignores that there are other sources of errors which could affect these sparse samples.

[1]Definitions of all of the metrics besides the MAE can be found in [2]. Mean Absolute Error (MAE) is the average pixel-wise absolute difference between the prediction and ground truth depth.

TABLE I: Prediction results on the simulated forest environment for a trained model with different methods for feature sampling. Entries less than 10% worse than the baseline are yellow, and entries that are at least 10% worse than the baseline are red. The network with the best metric is bolded per-section.

Test	Train	Model	$\delta_1 \uparrow$	$\delta_2 \uparrow$	$\delta_3 \uparrow$	RMSE ↓	MAE ↓	$\log_{10} \downarrow$
-	-	RGB	0.552	0.758	0.859	6.409	3.420	0.144
uniform	unif	RGBsD	0.704	0.835	0.896	6.401	2.902	0.110
		RGBsD-late	**0.713**	**0.844**	**0.904**	**5.996**	**2.737**	**0.104**
	fast	RGBsD	0.664	0.819	0.888	6.560	3.087	0.120
		RGBsD-late	0.687	0.831	0.895	6.365	3.011	0.115
	robust	RGBsD	0.629	0.803	0.878	6.654	3.260	0.133
		RGBsD-late	0.668	0.821	0.889	6.479	3.036	0.118
corners	unif	RGBsD	0.522	0.718	0.821	6.999	4.010	0.168
		RGBsD-late	0.528	0.724	0.825	6.935	4.086	0.166
	fast	RGBsD	0.613	0.790	0.873	6.281	3.258	0.133
		RGBsD-late	**0.641**	**0.810**	**0.886**	**6.054**	**3.065**	**0.123**
	robust	RGBsD	0.595	0.791	0.875	6.369	3.328	0.137
		RGBsD-late	0.624	0.801	0.881	6.182	3.123	0.127
noisy corners	unif	RGBsD	0.443	0.651	0.766	8.025	4.749	0.207
		RGBsD-late	0.462	0.672	0.787	7.688	4.688	0.192
	fast	RGBsD	0.537	0.738	0.835	7.061	3.844	0.160
		RGBsD-late	0.590	0.781	0.867	6.619	3.455	0.138
	robust	RGBsD	0.572	0.779	0.867	6.594	3.458	0.143
		RGBsD-late	**0.613**	**0.794**	**0.877**	**6.308**	**3.192**	**0.130**

3) Robustness to Noisy Sparse Inputs

To investigate the robustness of the networks to the noises from imperfect VIO depths and bearings as well as the artifacts from projecting into the current frame with imperfect relative pose with respect to the feature's anchor frame, we propose to simulate the noisy sparse point input from VIO as follows. For each FAST corner location $\mathbf{u} = [u \; v]^\top$, we first extract the groundtruth depth value d and perturb the coordinates and depth separately with a zero mean Gaussian with standard deviation of 3 pixels and 0.45 meters for the depths, respectively. For the camera pose error, we take the perturbed coordinates and depth, project it into the camera frame , and then apply a randomly generated SE(3) transformation to the camera pose with standard deviation 3° and 3cm, after which the points are re-projected back into the image plane. From Table I, we see that the (robust) networks trained with the simulated noises can also remain robust to the noises during testing with unseen data, while most of the other methods are unable to outperform the RGB network metrics.

4) Sensitivity to Feature Density

The sparse feature density can vary widely in a VIO system due to conditions such as low image gradients, fast motion, limited computational resources, or even cases were the VIO fails and state estimation is lost. Most works have only compared RGBsD networks to sparse depth-only networks, and not even considered how it compares to RGB only. We here focus on making the RGBsD network better than the RGB network for all possible sparsities of the VIO system.

Figure 2 shows the impact of varying the number of sparse features used during test time. Shown in the left column, the networks that were trained with a fixed 500 points perform worse than the RGB baseline when presented with sparsity that is even slightly less than that given at training time, but improve for higher densities – which is definitely desirable. Following [22], we decide to not include a validity mask for the sparse input and instead just allow the network

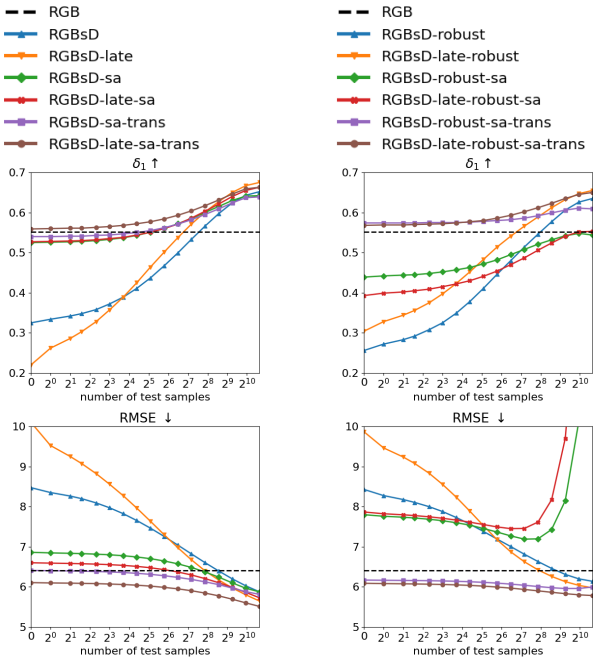


Fig. 2: All RGBsD networks here have been trained with FAST corners. Note that no perturbations were performed during test time to highlight the affects of varying sparsity. **Left:** Networks trained to be sparsity agnostic (*-sa), while ones with initialized weights from the RGB network (*-trans) are evaluated. **Right:** Networks trained with perturbations (*-robust) to groundtruth sparse points.

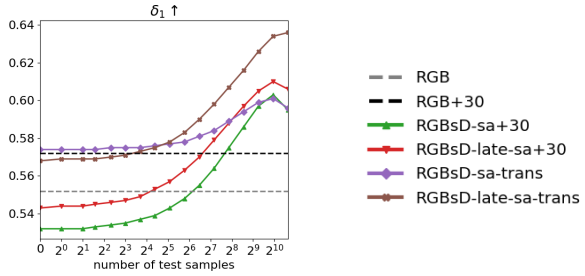


Fig. 3: Networks trained for additional 30 epochs (*+30) vs. the original networks with transferred weights from the RGB network.

to learn its own sparsity-agnostic features (marked as "sa" in Figure 2). We train these networks with 0-500 points chosen randomly and uniformly. It can be seen in Figure 2 (left column) that the networks are able to learn a natural invariance to the sparsity level, but are still generally worse than the RGB-only network at extreme sparsity levels. To address this issue, we initialize the depth completion network weights with a fully pre-trained RGB network for layers of compatible dimensions (including the sparse depth encoder of the late fusion model) and then train for the same 30 epochs with varying sparsity – which is marked as "trans" in Figure 2. This provides the best performance and ensures that even in cases where no points are present, we generally perform better than the RGB baseline. This method is highly intuitive, as the RGB network has already learned to handle 0% sparsity and the performance gain is still mostly retained when more points are supplied.

To see how the networks handle both varying sparsity and the noise perturbations during training, we perform the same analysis while training with both variations (see Figure 2 right column). Note that we do not apply any random

TABLE II: Validation of our proposes depth completion models with sparse depth inputs generated from our VIO system. Evaluation of the VIO sparse feature error is shown in the first row, and again the results are highlighed in yellow in red as they become worse than the baseline RGB network.

Train	Model	$\delta_1 \uparrow$	$\delta_2 \uparrow$	$\delta_3 \uparrow$	RMSE $\downarrow$	MAE $\downarrow$	$\log_{10} \downarrow$
-	VIO (sparse)	0.651	0.790	0.848	6.780	3.179	0.126
-	RGB	0.604	0.841	0.913	5.838	3.332	0.125
unif	RGBsD	0.431	0.674	0.802	8.736	5.083	0.188
	RGBsD-late	0.539	0.748	0.830	6.390	3.434	0.147
fast	RGBsD	0.436	0.694	0.830	7.817	4.535	0.175
	RGBsD-late	0.435	0.744	0.853	7.392	4.342	0.167
robust	RGBsD	0.593	0.771	0.858	7.357	3.917	0.142
	RGBsD-late	0.523	0.787	0.866	7.215	3.944	0.149
sa	RGBsD	0.445	0.774	0.874	7.058	4.120	0.155
	RGBsD-late	0.570	0.774	0.867	7.287	3.998	0.141
trans	RGBsD	0.582	**0.852**	**0.919**	**5.652**	3.318	0.127
	RGBsD-late	**0.658**	0.849	0.915	5.742	**3.211**	**0.122**

perturbations during testing in order to highlight the affect of the varying sparsity only. Here the benefit of transferring the knowledge from the RGB model becomes more clear, as the "sa" networks which have been been initialized with only the ImageNet weights perform poorly, and completely fail to converge in 30 epochs when trained with both the varying sparsity and the sparse depth perturbations. This shows the difficulty networks have in learning to predict depths from RGB while filtering out the noisy depth inputs which vanish randomly. Figure 3 shows that with an additional 30 epochs of training for all but the "trans" networks, and additionally introducing noisy points in testing again, the proposed training method still still yields better δ_1 accuracy – especially at extreme sparsity levels, verifying our findings.

5) Impact of VIO Sparse Point Inputs

To evaluate the performance of the network when directly leveraging sparse points generated from our VIO system, we collect a visual-inertial dataset of a small MAV flying through the forest along with groundtruth depthmaps within the simulator. Shown in Figure 1 and quantified in Table II, we evaluate the network prediction results when given these noisy VIO point inputs. Note first that the sparse depths produced by VIO are more noisy as compared to the RGB network on this dataset – while remaining generally more accurate as measured by the δ_1 metric, which is interesting – opposite of that for indoor sequences shown in [15], where the sparse depths were *more* accurate. This suggests the presence of some outliers in the VIO depths, especially at farther distances, in this challenging outdoor dataset. Despite this, we can see that the proposed network which is initialized with weights from the RGB and trained with noisy/varying points is generally able to outperform the RGB network and in some cases even the VIO points themselves. The high noise level of the VIO points additionally suggests that noisier inputs should be used in training, however the noises we use still prove beneficial, and we did not want to seemingly bias results with abnormally large simulated noises.

6) Summary

Training the network with perturbations to the depths, corner locations, and camera pose is crucial to allow for

TABLE III: Evaluation of existing methods and the proposed network under different sparse point selection criteria on the NYU Depth V2 dataset.

Test	Model	$\delta_1 \uparrow$	$\delta_2 \uparrow$	$\delta_3 \uparrow$	RMSE $\downarrow$	MAE $\downarrow$	abs. rel. $\downarrow$	$\log_{10} \downarrow$
-	FastDepth	0.904	0.971	0.989	0.433	0.267	0.092	0.042
uniform	S2D	**0.981**	**0.995**	**0.998**	**0.194**	**0.100**	**0.036**	**0.016**
	RGBsD	0.956	0.991	0.997	0.281	0.173	0.063	0.028
	RGBsD-late	0.963	0.991	0.997	0.270	0.160	0.057	0.025
corners	S2D	0.626	0.689	0.730	1.247	0.777	0.267	0.199
	RGBsD	0.948	0.990	0.997	0.301	0.194	0.072	0.032
	RGBsD-late	**0.955**	**0.990**	**0.997**	**0.286**	**0.176**	**0.064**	**0.028**
noisy corners	S2D	0.480	0.613	0.680	1.371	0.980	0.350	0.241
	RGBsD	0.942	0.988	0.996	0.316	0.203	0.075	0.032
	RGBsD-late	**0.950**	**0.989**	**0.996**	**0.298**	**0.186**	**0.069**	**0.030**

the network to robustly handle imperfect features. To handle varying levels of feature sparsity ranging from zero to a semi-dense image, initializing the network with weights from the RGB network and training with varying levels of sparsity ensures that the network never regresses worse than the baseline RGB network. While the late fusion (RGBsD-late) outperforms the shared encoder (RGBsD) in terms of accuracy, in Section IV-B this dual-encoder network clearly has a tradeoff in terms of prediction speed.

IV. Experimental Results

A. NYU Depth V2 Dataset

We choose the NYUv2 indoor dataset for benchmark experiments [36], which contains 407,024 RGBD images collected with a Kinect. The training images contain missing depth values due to occlusions and specular surfaces that the Kinect's depth sensor cannot handle. They additionally provide 1449 image depth pairs that have full-dense depth labels. We utilize 101,664 of the training samples and the full 1449 densely-labeled dataset for testing. We compare our networks to the original FastDepth [6] and the ResNet-based sparse-to-dense (S2D) model [10] with RGB and sparse depth inputs. The S2D model is initialized with ResNet-50 weights and trained with 500 uniformly-sampled depth values with no perturbation to the sparse depths as originally proposed in their work, and all networks are trained with the same configurations as described in Section III-B.1. Note that our proposed networks in this experiment are the same as the "trans" networks from before – initialized with RGB weights and trained with the sparse depth perturbations and varying sparsity as before.

Shown in Table III, the S2D ResNet model achieves higher accuracy than the proposed networks when the sparse points are drawn uniformly during test time. This is expected since it was trained on uniform depth values and contains many more parameters. However, when predicting with sparse points selected through true or noisy FAST corners, the S2D accuracy quickly degrades worse than the proposed networks and baseline RGB FastDepth. In fact, with noisy corners it predicts nearly unusable results with less than 50% δ_1 accuracy on this relatively easy dataset. This further validates our previous analysis that the difference in the testing distribution of sparse depths can easily degrade the performance far below that of the RGB network, even with the more powerful ResNet architecture. Here we stress that the proposed networks, which were initialized to RGB-only weights and trained with noisy and varying number of sparse depths, are robust over all variations and display *consistent* and *robust* accuracy levels.

B. On-Platform Depth Completion

We evaluate the proposed depth completion networks on a Jetson TX2 and Jetson Nano[2] device. Both are small, light-weight, and low-power platforms which have an on-board embedded GPU for network acceleration. The TX2 module has 256 Pascal CUDA cores, a dual-core NVIDIA Denver 2 64-bit CPU, quad-core ARM A57 Complex and a max power of 15W. The Nano has 128 Maxwell CUDA cores with a quad-core ARM Cortex-A57 MPCore processor and a max power of 10W. Both systems are flashed with Jetpack 4.4 and evaluated using Max-N performance mode with jetson_clocks enabled. We leverage Apache TVM[3] as in [6] – with 5 trials of 200 network predictions each.

The results are shown in Table IV, where we evaluate the base FastDepth RGB network along with the two variations used in previous experiments. The RGBsD network has prediction speed at the same level as the RGB – showing that including the extra channel to gain in accuracy from sparse depth does not cause large impacts on latency for the embedded device. Interestingly enough, the 4-channel encoder is sometimes even faster than the 3-channel – which we suspect has to do with better vectorization and saturation of the CUDA thread and block allocations due to operations now being on a power of 2 elements. The RGBsD-late on the other hand seems to be affected by the constrained resources on these platforms with a large performance gap compared to the single encoder networks. However, the accuracy gains from the previous experiments thus expose a nice tradeoff between the two architectures, with the RGBsD providing a high frame-rate with slight loss of accuracy (e.g., in high-speed obstacle avoidance) and late fusion providing the best choice in the opposite case (e.g., for dense mapping). Even with its smaller number of CUDA cores and older architecture, the Jetson Nano still can provide satisfactory prediction speeds at 15fps on the CPU (nearly real-time) and over 50fps on the GPU. The performance gain from TVM tuning is non-trivial and thus is recommended to be performed.

Additionally, to obtain a direct comparison to popular non-learning methods, we provide timing results for traditional stereo depth methods to directly compare their speed to that of the networks (see Table V). All input images have already been pre-rectified, which is often not the case in practice, thus increasing the amount of time required beyond what is showed here. Both block matching (BM) [37] and semi-global block matching (SGBM) [38] leverage a block size of 5x5 with max disparity of 96 within the OpenCV library [39]. Both methods perform left-to-right and right-to-left disparity which are then combined using the post-processing Fast Global Smoother [40]. It is clear that traditional BM method can provide a high enough frequency on both the CPU and GPU, while the semi-global method falls slightly below real-time performance. In both cases however, the 3 and 4-channel tuned networks are generally faster – with

[2] *https://developer.nvidia.com/embedded/jetson-modules*
[3] *https://github.com/apache/incubator-tvm/*

TABLE IV: Timing results of the proposed network on different platforms. Average over 1000 network prediction, times in milliseconds, and input images are 224x224. Best in the group (green) and second best (blue) are highlighted.

Platform	Model	No-Optimization			TVM-Optimization		
		t_{pred}	σ	Hz	t_{pred}	σ	Hz
Nano (CPU)	RGB	88.87	12.55	11	66.54	14.31	15
	RGBsD	89.48	12.62	11	66.41	13.65	15
	RGBsD-late	140.54	3.88	7	112.18	11.37	9
Nano (GPU)	RGB	24.75	0.69	40	14.90	0.79	67
	RGBsD	24.91	0.58	40	17.07	1.03	58
	RGBsD-late	42.99	0.46	23	70.66	0.34	14
TX2 (CPU)	RGB	118.06	9.62	8	117.45	10.82	8
	RGBsD	118.45	9.52	8	105.07	13.01	9
	RGBsD-late	184.37	13.87	5	165.69	13.46	6
TX2 (GPU)	RGB	10.79	0.46	92	6.87	0.70	145
	RGBsD	10.78	0.60	92	7.09	0.71	141
	RGBsD-late	18.36	0.47	54	11.34	0.90	88

TABLE V: Traditional stereo disparity block matching methods on different platforms. Average over 1000 pre-rectified images, times in milliseconds, and images are 224x224.

Platform	Method	t_{disp}	σ	Hz
Nano (CPU)	BM	27.76	1.90	36
	SGBM	91.22	1.22	11
Nano (GPU)	BM	19.24	2.77	52
TX2 (CPU)	BM	19.49	0.40	51
	SGBM	64.29	0.32	15
TX2 (GPU)	BM	12.38	1.27	80

the exception of the TX2 CPU.[4] In contrast, the previously presented networks rely only on a single camera and can be trained to predict depths on distorted images – negating the extra cost of rectification and allowing execution on wider-variety resource-constrained devices.

C. On-Platform VIO

We now evaluate the the performance of our VIO system. By default, OpenVINS does not have any multi-threading nor GPU acceleration. We perform our on-device evaluation on the UZH-FPV drone racing dataset [41] which exhibits both the indoor / outdoor environments with high optical flow rate typically seen from a high speed MAV platform. This dataset is challenging due to 13-23 m/s top speeds, varying levels of motion blur, exposure changes, and low / repeated textures. For comparison to a typical "desktop" non-embedded system, we run the estimator on a computer which has an Intel(R) Xeon(R) CPU E3-1505M v6 @ 3.00GHz. Since different platforms require tuning of different estimator parameters to enable real-time performance, we evaluate both the accuracy and frame processing time. As before, all Jetson systems are run in Max-N mode. The key estimator settings used for each device are as follows:

- *Jetson Nano:* 125 features tracked, max of 10 MSCKF update per-frame, max of 15 SLAM features.
- *Jetson TX2:* 175 features tracked, max of 15 MSCKF update per-frame, max of 15 SLAM features.
- *Desktop:* 600 features tracked, max of 20 MSCKF update per-frame, max of 15 SLAM features.

[4]In fact, in both cases (GPU and CPU), our timings are slightly worse, but on the same magnitude, of those reported in [6] (especially in the CPU case), which we suspect is due to using different tuning parameters.

TABLE VI: Relative pose error, degree per meter and percent translation, on the competition dataset along with average amount of time (in seconds) to perform a frame update and its standard deviation. All errors are based on the results reported by the online tool which averages errors over the 40m, 60m, 80m, 100m, 120m sub-trajectory lengths.

Dataset	Platform	Ori.	Pos.	t_{track}	t_{total}	σ_{total}
indoor forward 11	Nano	0.6079	7.474	0.0192	0.0359	0.0115
	TX2	0.6075	7.432	0.0128	0.0280	0.0088
	Desktop	0.6074	7.416	0.0085	0.0183	0.0044
indoor 45° 3	Nano	0.2507	2.675	0.0198	0.0391	0.0119
	TX2	0.2503	2.700	0.0131	0.0304	0.0097
	Desktop	0.2484	2.649	0.0088	0.0191	0.0048
outdoor forward 9	Nano	0.1762	7.994	0.0190	0.0339	0.0134
	TX2	0.1780	7.839	0.0126	0.0265	0.0103
	Desktop	0.1804	7.832	0.0084	0.0175	0.0056

We report the results on a series of datasets in Table VI. In general all devices achieve the same order of estimation errors. It can also be seen that even with the Jetson Nano only tracking 125 features, it has almost double the tracking processing time, t_{track}, as the desktop system due to its lower CPU rate. In general the embedded systems can process in real time at 28fps or 0.0357 seconds, and thus, it is amenable to use OpenVINS as the sparse depth supplier.

V. Conclusions and Future Work

In this work we have investigated the use of depth completion networks to generate dense depth maps from noisy sparse depth inputs from an external VIO system. To this end, we have tailored our OpenVINS to provide better inputs into the depth completion network. As naïve training of the depth completion network with true uniformly-sampled depths results in significantly worst accuracy than the original image-only network, we have investigated different training schemes to address these sensitivities. We proposed to initialize from the image-only weights and train with extremely noisy data, showing large gains in robustness while generally outperforming the RGB network – even with wildly varying sparsities and noisy inputs. Finally, we have performed a thorough computational analysis on the Jetson Nano and TX2 embedded platforms, showing real-time performance of both our state estimator and depth completion network compare to traditional methods. In the future, we plan to deploy this work in a MAV path planning system, and include both high frame rate depths for obstacle avoidance as well as asynchronous dense mapping for long term navigation.

VI. Acknowledgements

This work was partially supported by the University of Delaware (UD) College of Engineering, the ARL (W911NF-19-2-0226), and Google ARCore. P. Geneva was also partially supported by the Delaware Space Grant College and Fellowship Program (NASA Grant NNX15AI19H). We additionally thank S. Nogar and the ARL ASD Aerial Vehicle Team for their simulation dataset generation support.

REFERENCES

[1] P. Geneva, K. Eckenhoff, W. Lee, Y. Yang, and G. Huang, "Openvins: A research platform for visual-inertial estimation," in *Proc. of the IEEE International Conference on Robotics and Automation*, Paris, France, 2020. [Online]. Available: https://github.com/rpng/open_vins

[2] D. Eigen, C. Puhrsch, and R. Fergus, "Depth map prediction from a single image using a multi-scale deep network," in *Advances in Neural Information Processing Systems 27*, 2014, pp. 2366–2374.

[3] I. Laina, C. Rupprecht, V. Belagiannis, F. Tombari, and N. Navab, "Deeper depth prediction with fully convolutional residual networks," in *3D Vision (3DV), 2016 Fourth International Conference on*. IEEE, 2016, pp. 239–248.

[4] H. Fu, M. Gong, C. Wang, K. Batmanghelich, and D. Tao, "Deep ordinal regression network for monocular depth estimation," in *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2018.

[5] Z. Li and N. Snavely, "Megadepth: Learning single-view depth prediction from internet photos," in *Computer Vision and Pattern Recognition (CVPR)*, 2018.

[6] D. Wofk, F. Ma, T.-J. Yang, S. Karaman, and V. Sze, "Fastdepth: Fast monocular depth estimation on embedded systems," in *IEEE International Conference on Robotics and Automation (ICRA)*, 2019.

[7] B. Ummenhofer, H. Zhou, J. Uhrig, N. Mayer, E. Ilg, A. Dosovitskiy, and T. Brox, "Demon: Depth and motion network for learning monocular stereo," in *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2017, pp. 5038–5047.

[8] H. Zhou, B. Ummenhofer, and T. Brox, "Deeptam: Deep tracking and mapping," in *Proceedings of the European conference on computer vision (ECCV)*, 2018, pp. 822–838.

[9] C. Liu, J. Gu, K. Kim, S. G. Narasimhan, and J. Kautz, "Neural rgbd sensing: Depth and uncertainty from a video camera," in *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2019, pp. 10 986–10 995.

[10] F. Ma and S. Karaman, "Sparse-to-dense: Depth prediction from sparse depth samples and a single image," in *IEEE International Conference on Robotics and Automation (ICRA)*, 2018.

[11] F. Ma, G. V. Cavalheiro, and S. Karaman, "Self-supervised sparse-to-dense: Self-supervised depth completion from lidar and monocular camera," in *IEEE International Conference on Robotics and Automation (ICRA)*, 2019.

[12] J. Qiu, Z. Cui, Y. Zhang, X. Zhang, S. Liu, B. Zeng, and M. Pollefeys, "Deeplidar: Deep surface normal guided depth prediction for outdoor scene from sparse lidar data and single color image," in *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2019.

[13] L. Teixeira, M. R. Oswald, M. Pollefeys, and M. Chli, "Aerial single-view depth completion with image-guided uncertainty estimation," *IEEE Robotics and Automation Letters (RA-L)*, pp. 1055–1062, 2020.

[14] A. Wong, X. Fei, S. Tsuei, and S. Soatto, "Unsupervised depth completion from visual inertial odometry," *IEEE Robotics and Automation Letters*, pp. 1899–1906, 2020.

[15] K. Sartipi, T. Do, T. Ke, K. Vuong, and S. Roumeliotis, "Deep depth estimation from visual-inertial slam," *ArXiv*, vol. abs/2008.00092, 2020.

[16] Y. Zhang and T. Funkhouser, "Deep depth completion of a single rgb-d image," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2018, pp. 175–185.

[17] A. Angelova, D. Yamparala, J. Vincent, and C. Leger, "Onboarddepth: Depth prediction for onboard systems," in *2019 European Conference on Mobile Robots (ECMR)*. IEEE, 2019, pp. 1–8.

[18] N. Chodosh, C. Wang, and S. Lucey, "Deep convolutional compressed sensing for lidar depth completion," in *Asian Conference on Computer Vision*. Springer, 2018, pp. 499–513.

[19] J. Uhrig, N. Schneider, L. Schneider, U. Franke, T. Brox, and A. Geiger, "Sparsity invariant cnns," in *2017 international conference on 3D Vision (3DV)*. IEEE, 2017, pp. 11–20.

[20] Y. Yang, A. Wong, and S. Soatto, "Dense depth posterior (ddp) from single image and sparse range," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2019, pp. 3353–3362.

[21] H. Jang, D. S. Jeon, H. Ha, and M. H. Kim, "Fast omnidirectional depth densification," in *International Symposium on Visual Computing*. Springer, 2019, pp. 683–694.

[22] M. Jaritz, R. De Charette, E. Wirbel, X. Perrotton, and F. Nashashibi, "Sparse and dense data with cnns: Depth completion and semantic segmentation," in *2018 International Conference on 3D Vision (3DV)*. IEEE, 2018, pp. 52–60.

[23] X. Cheng, Y. Zhong, Y. Dai, P. Ji, and H. Li, "Noise-aware unsupervised deep lidar-stereo fusion," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2019, pp. 6339–6348.

[24] S. S. Shivakumar, T. Nguyen, I. D. Miller, S. W. Chen, V. Kumar, and C. J. Taylor, "Dfusenet: Deep fusion of rgb and sparse depth information for image guided dense depth completion," in *2019 IEEE Intelligent Transportation Systems Conference (ITSC)*. IEEE, 2019, pp. 13–20.

[25] A. Eldesokey, M. Felsberg, and F. S. Khan, "Confidence propagation through cnns for guided sparse depth regression," *IEEE transactions on pattern analysis and machine intelligence*, 2019.

[26] J. Hua and X. Gong, "A normalized convolutional neural network for guided sparse depth upsampling." in *Proceedings of the Twenty-Seventh International Joint Conference on Artificial Intelligence*, 2018, pp. 2283–2290.

[27] Y. Xu, X. Zhu, J. Shi, G. Zhang, H. Bao, and H. Li, "Depth completion from sparse lidar data with depth-normal constraints," in *Proceedings of the IEEE/CVF International Conference on Computer Vision*, 2019, pp. 2811–2820.

[28] A. Lopez-Rodriguez, B. Busam, and K. Mikolajczyk, "Project to adapt: Domain adaptation for depth completion from noisy and sparse sensor data," *arXiv preprint arXiv:2008.01034*, 2020.

[29] A. Rosinol, T. Sattler, M. Pollefeys, and L. Carlone, "Incremental visual-inertial 3d mesh generation with structural regularities," in *2019 International Conference on Robotics and Automation (ICRA)*. IEEE, 2019, pp. 8220–8226.

[30] A. I. Mourikis and S. I. Roumeliotis, "A multi-state constraint kalman filter for vision-aided inertial navigation," in *Proceedings 2007 IEEE International Conference on Robotics and Automation*. IEEE, 2007, pp. 3565–3572.

[31] M. Li and A. I. Mourikis, "Optimization-based estimator design for vision-aided inertial navigation," in *Robotics: Science and Systems*. Berlin Germany, 2013, pp. 241–248.

[32] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, A. Desmaison, A. Kopf, E. Yang, Z. DeVito, M. Raison, A. Tejani, S. Chilamkurthy, B. Steiner, L. Fang, J. Bai, and S. Chintala, "Pytorch: An imperative style, high-performance deep learning library," in *Advances in Neural Information Processing Systems 32*, 2019, pp. 8024–8035.

[33] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," *arXiv preprint arXiv:1412.6980*, 2014.

[34] A. G. Howard, M. Zhu, B. Chen, D. Kalenichenko, W. Wang, T. Weyand, M. Andreetto, and H. Adam, "Mobilenets: Efficient convolutional neural networks for mobile vision applications," *arXiv preprint arXiv:1704.04861*, 2017.

[35] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei, "Imagenet: A large-scale hierarchical image database," in *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2009, pp. 248–255.

[36] N. Silberman, D. Hoiem, P. Kohli, and R. Fergus, "Indoor segmentation and support inference from rgbd images," in *European conference on computer vision*. Springer, 2012, pp. 746–760.

[37] K. Konolige, "Projected texture stereo," in *2010 IEEE International Conference on Robotics and Automation*. IEEE, 2010, pp. 148–155.

[38] H. Hirschmuller, "Stereo processing by semiglobal matching and mutual information," *IEEE Transactions on pattern analysis and machine intelligence*, vol. 30, no. 2, pp. 328–341, 2007.

[39] R. Szeliski, *Computer vision: algorithms and applications*. Springer Science & Business Media, 2010.

[40] D. Min, S. Choi, J. Lu, B. Ham, K. Sohn, and M. N. Do, "Fast global image smoothing based on weighted least squares," *IEEE Transactions on Image Processing*, vol. 23, no. 12, pp. 5638–5653, 2014.

[41] J. Delmerico, T. Cieslewski, H. Rebecq, M. Faessler, and D. Scaramuzza, "Are we ready for autonomous drone racing? the uzhfpv drone racing dataset," in *IEEE International Conference on Robotics and Automation (ICRA)*, 2019.