Metrically-Scaled Monocular SLAM using Learned Scale Factors

W. Nicholas Greene Nicholas Roy

***Abstract*— We propose an efficient method for monocular simultaneous localization and mapping (SLAM) that is capable of estimating metrically-scaled motion without additional sensors or hardware acceleration by integrating metric depth predictions from a neural network into a geometric SLAM factor graph. Unlike learned end-to-end SLAM systems, ours does not ignore the relative geometry directly observable in the images. Unlike existing learned depth estimation approaches, ours leverages the insight that when used to estimate scale, learned depth predictions need only be coarse in image space. This allows us to shrink our network to the point that performing inference on a standard CPU becomes computationally tractable.**

We make several improvements to our network architecture and training procedure to address the lack of depth observability when using coarse images, which allows us to estimate spatially coarse, but depth-accurate predictions in only 30 ms per frame without GPU acceleration. At runtime we incorporate the learned metric data as unary scale factors in a Sim(3) pose graph. Our method is able to generate accurate, scaled poses without additional sensors, hardware accelerators, or special maneuvers and does not ignore or corrupt the observable epipolar geometry. We show compelling results on the KITTI benchmark dataset in addition to real-world experiments with a handheld camera.

I. Introduction

The field of monocular simultaneous localization and mapping (SLAM), in which both egomotion and environmental structure are estimated from a single moving camera, has undergone tremendous advances over the past twenty years. Early filter-based approaches [2] have quickly evolved to sophisticated, hierarchical, factor graph-based optimizations, such as the methods of [3]–[7].

Due to the projective nature of cameras, however, these monocular systems – which rely solely on the geometric content of the images – can only estimate camera egomotion and environmental structure up to an arbitrary scale factor. Additional information must be exploited to resolve the metric scale of the solution. While leveraging priors over the camera's altitude or the size of known objects in the scene [8] is possible, the current most popular technique is to fuse the images with an inertial measurement unit (IMU), which measures linear accelerations and angular velocities at metric scale.

Though significant progress has been made on this front [9]–[13], these existing visual-inertial SLAM approaches have a number of drawbacks. Beyond the additional hardware that must be calibrated and time-synchronized, these algorithms are difficult to implement, often require expert parameter tuning, are extremely sensitive to errors in the accelerometer biases, and require high-acceleration motion to excite the IMU and make scale observable. (The last point is particularly troublesome for mobile robot navigation as it can significantly complicate the motion planning problem.)

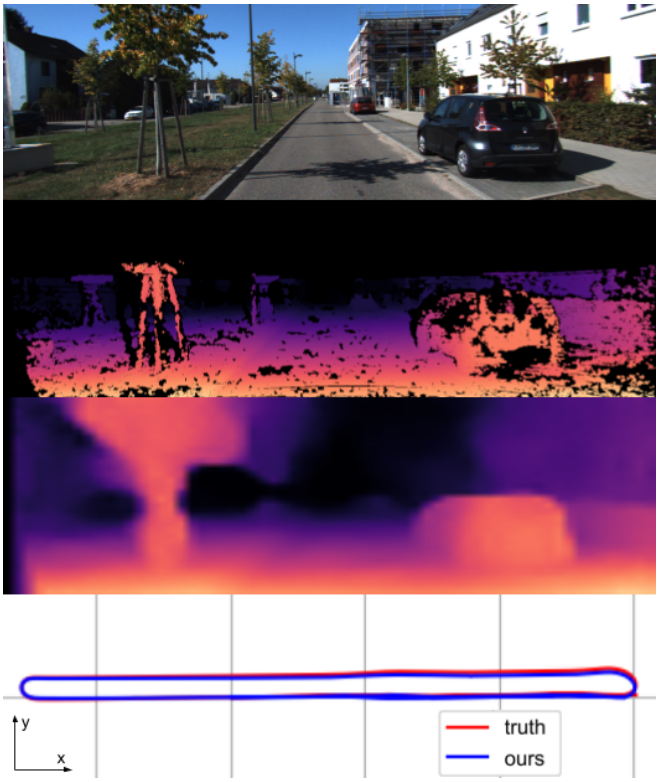

Fig. 1: *Metric Monocular SLAM:* Our method is capable of estimating metric camera motion from monocular images without additional sensors or hardware acceleration by leveraging depth predictions from a small neural network. *Top row*: Input image from the KITTI dataset [1]. *Second row*: Groundtruth depths from LIDAR scans. *Third row*: Coarse depthmap predicted with our network. *Bottom row*: Resulting metrically scaled trajectory (blue) versus the groundtruth (red).

To address these limitations, methods that apply end-to-end deep learning techniques to the monocular SLAM problem have appeared in recent years, spurred by the rapid adoption of deep, convolutional neural networks (CNNs) for a variety of computer vision tasks [14]–[18]. Although these end-to-end SLAM systems show promise, and can output scaled solutions, their tracking performance still lags behind geometric approaches and they require GPU acceleration to perform inference.

Computer Science and Artificial Intelligence Laboratory, Massachusetts Institute of Technology, Cambridge, MA 02139 {wng,nickroy}@csail.mit.edu

This material is based upon work supported by the NSF Graduate Research Fellowship under Grant Number 1122374 and by the Army Research Laboratory under Cooperative Agreement Number W911NF-17-2-0181. Their support is gratefully acknowledged.

Given that the relative geometry of the monocular SLAM problem is directly observable from the image data, we believe a more targeted application of machine learning (which does not ignore directly observable quantities in the sensor data) will ultimately lead to more robust systems.

To that end, we propose a monocular SLAM solution that combines metric information that can be inferred using a neural network with the state-of-the-art in factor graph-based geometric SLAM. We first train a small CNN to regress metric depth from monocular images given calibrated stereo frames as training data. Unlike existing learned depth estimation approaches [19]–[21], our technique leverages the insight that when used to estimate scale, these learned predictions need only be coarse in image space. This allows us to shrink our network to the point that performing inference on a standard CPU becomes computationally tractable. Simply downsampling the input images and training the network to minimize photoconsistency between stereo training pairs yields inaccurate depths, however, as the disparity between the left and right images decreases with image resolution. We make several improvements to our network architecture and training procedure to address this lack of depth observability, while keeping the efficiency that comes with using coarse input images.

First, although coarse images are used as input to the network at test time, we train on full resolution images and compute additional photoconsistency loss terms in a fine-to-coarse manner. Incorporating these loss terms at training time means that photoconsistency errors that are only observable at fine image scales can still be used to learn the disparity at the coarser image scales that we care about.

Second, we provide an additional supervision signal to the network by estimating a full-resolution disparity map using conventional block-matching stereo. Although these directly-computed disparity maps can be sparse and noisy, they nonetheless provide a loss signal that can allow the network to learn the correct disparity values at coarse image scales where photoconsistency may be insufficient.

These improvements allow us to estimate spatially coarse, but depth-accurate predictions in only 30 ms per frame on a standard CPU. At runtime we divide the SLAM problem into a local visual odometry (VO) module and a global pose graph module (see Figure 3). The local VO module performs conventional monocular SLAM over a small sliding window of keyframes, while the global pose graph incorporates metric depth measurements from the network to constrain the solution scale. After each iteration of solving for the camera poses and landmark positions, the scale of the global pose graph can be used to warp the local VO so that metrically scaled geometry is available for the most recent image.

Our method has notable advantages over existing approaches. Unlike inertial-based systems, we do not require extra sensors or special motion to generated scaled outputs. Unlike end-to-end learning-based systems, we do not ignore the observable epipolar geometry present in the live images and can take advantage of factor graph optimization. Unlike learned monocular depth estimation methods, we target the network specifically for scale estimation and can thus shrink the network to allow for fast inference without hardware acceleration. We show compelling results on the KITTI benchmark dataset in addition to real-world experiments with a handheld camera.

II. Related Work

Applying machine learning to solve aspects of the monocular SLAM problem has seen a recent resurgence in the literature due to the increasing expressive power of deep neural networks. While some methods target the monocular visual odometry problem specifically [16], [17], single-view depth estimation has seen an explosion of progress in the last decade. Initial methods used explicit supervision from ground truth models or LIDAR scans to regress depth from images [22], [23], while more recent approaches use self-supervision in the form of calibrated stereo imagery in order to regress depth [19], [21]. These self-supervised networks are increasingly augmented with separate pose estimation modules so that they can be applied directly to monocular video instead of calibrated stereo imagery [14], [15], [18], [20].

Our approach is most similar to the hybrid learned/geometric methods of [24]–[26], which combine learned priors with geometric SLAM. CNN-SLAM [24] uses a CNN to predict a depthmap for each frame in a keyframe SLAM graph, which is then iteratively refined and fused into a global map. DPC-Net [25] trains a network to provide *corrections* to an existing visual odometry pipeline. DVSO [26] predicts a hypothetical stereo image from a monocular image and then uses the pair of images to drive a stereo visual odometry system [27].

III. Method

A. Notation

We represent the image taken at time k by the function $I_k : \Omega \to \mathbb{R}$ over the pixel domain $\Omega \subset \mathbb{R}^2$. $\mathbf{K} \in \mathbb{R}^{3\times3}$ denotes the intrinsic camera parameters. We represent the pose of the camera at time k relative to frame j by $\mathbf{T}_k^j \in \mathbb{SE}(3)$. An element of the group of 3D similarity transforms $\mathrm{Sim}(3)$ is denoted by $\mathbf{S}_k^j$. The perspective projection function is denoted by $\pi(x, y, z) = (x/z, y/z)$. Vectors represented in homogeneous coordinates are denoted by $\bar{\mathbf{x}} = (\mathbf{x}, 1) \in \mathbb{R}^{n+1}$ for $\mathbf{x} \in \mathbb{R}^n$.

Given rectified stereo images, we let $D_l : \Omega \to \mathbb{R}$ represent the disparity map that warps the right image I_r to the left image I_l such that $I_l(u) = I_r(u + D_l(u))$. Similarly we let D_r represent the disparity map that warps the left image to the right image. A disparity map D can be converted to inverse depthmap Z given the horizontal focal length f_x of the cameras and the horizontal baseline B as $Z(u) = D(u)/(Bf_x)$.

B. Single-view Depth Regression

Following the self-supervised approach of Godard et al. [19], we estimate the metric inverse depthmap Z for a given image by treating it as the left image I_l of a calibrated

stereo pair and training a CNN to predict the corresponding right image I_r via the disparity maps D_l and D_r. We therefore wish to learn a function f with parameters θ that maps I_l to I_r (and vice versa) via D_l and D_r:

$$\begin{aligned}(D_l, D_r) &= f(I_l; \theta) \\ I_l(u) &= I_r(u + D_l(u)) \\ I_r(u) &= I_l(u + D_r(u)).\end{aligned} \tag{1}$$

With a dataset of stereo pairs $\mathcal{D} = \{I_l, I_r\}_j$ and a loss function l that measures the quality of the predictions, we can estimate the parameters θ by solving the following optimization problem:

$$\theta^* = \arg\min_{\theta} \sum_{I_l, I_r \in \mathcal{D}} l(f(I_l; \theta), I_l, I_r). \tag{2}$$

1) Network Architecture: We choose f to be a convolutional neural network for the power of these models to capture complex patterns in image data, while still being practical to train. Specifically, we base our network on the pyramidal model detailed by Poggi et al. [21] augmented with residual blocks [28]. This network significantly reduces the number of parameters required to regress disparity compared to the seminal approaches of [15], [18], [19]. Since we are interested in scale estimation, however, we can further simplify the network architecture. The model is built using three main building blocks: a feature extractor block, a disparity estimator block, and an upsampler block repeated at multiple image scales as shown in Figure 2.

The feature extractor block is built using four 3×3 convolutional layers with ReLU activations [29] arranged with skip connections into two residual blocks [28]. The first convolutional layer in the block also performs downsampling with a stride of 2. The number of filters depends on the image scale. The disparity estimator block is composed of a series of four 3×3 residual layers, with the first three using ReLU activations and the final output layer using a sigmoid activation to ensure positive disparities. The number of filters per layer in this block is 96, 64, 32, and 8, respectively. The upsampler block is simply a transpose convolution with stride 2 with the same number of filters as the input.

Given an input image of a particular resolution, we define 7 pyramid levels of interest: $L0$ (the base image) through $L6$ (the coarsest resolution). To ensure computational efficiency on constrained hardware, we only extract features from $L3$ to $L6$ by stacking feature extractor blocks with 16, 32, 64, and 128 filters at each respective scale. At $L6$, we attach a disparity estimator block directly to the feature extractor outputs to yield D^6 – the disparity map for the coarsest image scale. For $L3$ to $L5$ we take the features from each scale and concatenate them with those from the next coarsest scale after passing them through an upsampler block. These concatenated features are then fed into a disparity estimator block to generate $D^3 \dots D^5$. The finest disparity maps $D^0 \dots D^2$ are generated by simple bilinear interpolation for efficiency. This simple model is expressive enough to learn high-quality (but coarse) disparity maps despite having only 2.3 million trainable parameters.

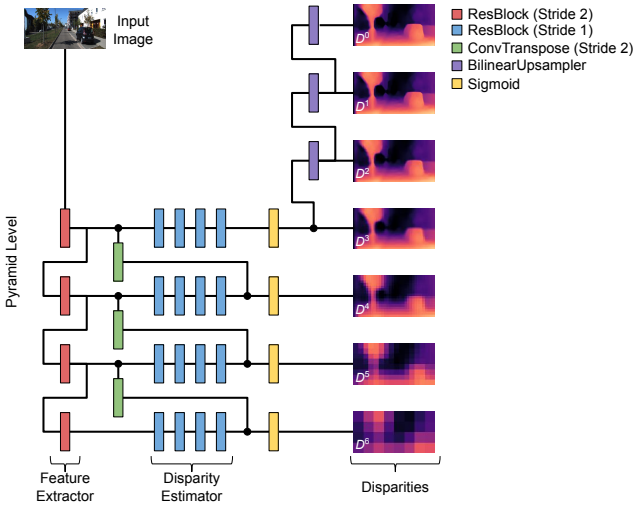

Fig. 2: *Network Architecture:* Our network follows the pyramidal structure of [21] with a series of feature extractor blocks and disparity estimator blocks at several image scales. Each processing block is comprised of residual blocks [28] with the number of filters varying depending on the pyramid level.

2) Loss Function: We train our network to regress disparities by minimizing a loss function composed of four terms: a photoconsistency loss l_p, a left-right consistency loss l_{lr}, a disparity regularization term l_r, and a supervision term l_s, defined at each image scale:

$$\begin{aligned}l(f(I_l;\theta), I_l, I_r) = \sum_{i=0}^{6} &\lambda_p \left(l_p(I_l^i, \hat{I}_l^i) + l_p(I_r^i, \hat{I}_r^i) \right) + \\ &\lambda_{lr} l_{lr}(D_l^i, D_r^i) + \\ &\lambda_r \left(l_r(D_l^i) + l_r(D_r^i) \right) \\ &\lambda_s \left(l_s(D_l^i) + l_s(D_r^i) \right).\end{aligned} \tag{3}$$

The photoconsistency term l_p measures the photometric error between the input image I and the image predicted using the estimated disparity maps $\hat{I}$. Following Godard et al. [19], we set l_p to be a combination of structural similarity SSIM and a simple L_1 error:

$$\begin{aligned}l_p(I, \hat{I}) = \frac{1}{N} \sum_{\mathbf{u} \in \Omega} &\alpha \frac{1 - \text{SSIM}(I(\mathbf{u}), \hat{I}(\mathbf{u}))}{2} + \\ &(1 - \alpha) |I(\mathbf{u}) - \hat{I}(\mathbf{u})|,\end{aligned} \tag{4}$$

where N is the number of pixels in the image at a given scale and $\alpha > 0$ controls the weighting between the SSIM and L_1 terms.

The left-right consistency loss l_{lr} measures the discrepancy between the left and right disparity maps after warping them into each other:

$$\begin{aligned}l_{lr}(D_l, D_r) = \frac{1}{N} \sum_{\mathbf{u} \in \Omega} &|D_l(\mathbf{u}) - D_r(\mathbf{u} + D_l(\mathbf{u}))| + \\ &|D_r(\mathbf{u}) - D_l(\mathbf{u} + D_r(\mathbf{u}))|.\end{aligned} \tag{5}$$

The disparity regularization term l_r penalizes non-smooth

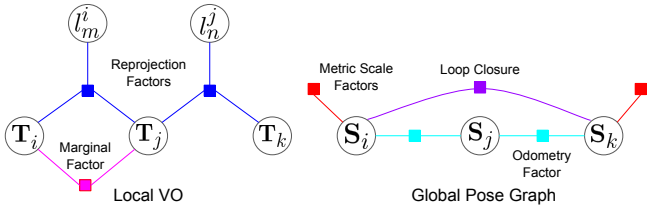

Fig. 3: *Factor Graphs:* Our monocular SLAM backend is composed of two factor graphs: a local visual odometry (VO) graph (left) and a global pose graph (right) The local VO module estimates unscaled camera poses and landmarks, while the global pose graph fuses marginalized keyframes from the local VO module with metric scale factors generated by our neural network.

disparity maps where the image gradient is low:

$$l_r(D) = \frac{1}{N}\sum_{\mathbf{u}\in\Omega} e^{-||\nabla_x I(\mathbf{u})||}\nabla_x D(\mathbf{u}) + e^{-||\nabla_y I(\mathbf{u})||}\nabla_y D(\mathbf{u}). \tag{6}$$

This loss is applied to both the left and right disparity maps.

The final supervision loss l_s measures the Huber error between the estimated disparity maps D_l and D_r and disparity maps generated using traditional block-matching B_l and B_r:

$$l_s(D) = \frac{1}{N}\sum_{\mathbf{u}\in\Omega} ||D(\mathbf{u}) - B(\mathbf{u})||_\epsilon, \tag{7}$$

where $\epsilon > 0$ is the parameter that governs when the Huber norm switches between squared and linear error.

C. Local Visual Odometry

Our local monocular VO pipeline is divided into a frontend module that builds a factor graph $\mathcal{G}_L = (\mathcal{V}_L, \mathcal{F}_L)$ from the raw image stream and a backend module that optimizes variables $\mathcal{V}_L$ (see Figure 3). $\mathcal{V}_L$ contains keyframe poses $\mathcal{K}_L$ and landmark map $\mathcal{M}_L$. The factor set $\mathcal{F}_L$ is composed of reprojection factors r_p that link keyframes and landmarks.

1) Frontend: At each new frame I_k, we detect corners using the method of [30] and track them from frame to frame using Lucas-Kanade [31], [32]. When the average pixel motion of the features between the last keyframe and current image exceeds a threshold, we create a new keyframe with pose $\mathbf{T}_k^W \in \mathbb{SE}(3)$, initialized by running motion-only Bundle Adjustment with respect to the existing map $\mathcal{M}_L = \{l_j^i\}$ comprised of landmarks l_j^i. Each landmark l_j^i is parameterized by its pixel location $\mathbf{u}_j \in \Omega$, its inverse depth $\xi_j \in \mathbb{R}^+$, and the frame it was detected in i. Features that were detected in the current frame k are initialized as new landmarks. Observations of pre-existing landmarks in k are added to $\mathcal{F}_L$ as reprojection factors.

2) Reprojection Factors: Each time a landmark l_j^i is observed in a new keyframe k, we insert a reprojection factor into the graph that constrains the landmark's inverse depth and the poses of the keyframes in which it was observed. Suppose that landmark l_j^i is observed in keyframe k at pixel location $\mathbf{p}_j \in \Omega$. The reprojection error r_p from this observation is given by

$$r_p(\mathbf{T}_i^W, \mathbf{T}_k^W, l_j^i) = \pi\left(\mathbf{K}\mathbf{T}_W^k\mathbf{T}_i^W\mathbf{K}^{-1}\bar{\mathbf{u}}_j/\xi_j\right) - \mathbf{p}_j. \tag{8}$$

3) Backend: After a new keyframe is initialized and all new factors are added to the graph, we enqueue a solve operation that will take place in a background thread. The total cost represented by $\mathcal{G}_L$ can be written as

$$E_L(\mathcal{K}_L, \mathcal{M}_L) = \sum_{i,j,k\in\mathcal{F}_L} ||r_p^{ijk}\left(\mathbf{T}_i^W, \mathbf{T}_j^W, l_k^i\right)||_\epsilon \tag{9}$$

where $||\cdot||_\epsilon$ represents the Huber norm with parameter $\epsilon > 0$.

This objective function is a (robust) sum of squared residuals, which we can optimize using the Levenberg-Maquardt algorithm [33], [34].

4) Marginalization: We marginalize out old keyframes to ensure real-time processing. Suppose we wish to marginalize out keyframe k and its child landmarks. We will denote this set of variables by $v_y = \{\mathbf{T}_k^W, \{l_j^k\}\}$. We then find the factors $\mathcal{F}_{sep}$ that connect v_y to $\mathcal{G}_L$. Let v_x denote the variables in $\mathcal{V}_L$ that are connected to $\mathcal{F}_{sep}$, but are not in v_y. The variables v_x and v_y and the factors $\mathcal{F}_{sep}$ form a subgraph $\mathcal{G}_{sep} \subset \mathcal{G}_L$. The cost associated with this subgraph is given by

$$E_{sep}(v_x, v_y) = \sum_{i,j,k\in\mathcal{F}_{sep}} ||r_p\left(\mathbf{T}_k^W, \mathbf{T}_j^W, l_k^i\right)||_\epsilon. \tag{10}$$

Linearizing r_p around the current estimates of v_x and v_y yields a quadratic cost in the tangent space of v_x and v_y. We can then eliminate the v_y component of the cost via the Schur complement, leaving a quadratic factor $\mathcal{F}_\delta$ on v_x.

With v_y eliminated, we remove the variables v_y and the factors $\mathcal{F}_{sep}$ from $\mathcal{G}_L$ and add the marginal factor $\mathcal{F}_\delta$ onto the remaining variables v_x. The marginalized keyframe $\mathbf{T}_k^W$ and landmarks l_j^k are then passed to the global pose graph (see Section III-D).

D. Global Pose Graph

Once a local keyframe k and its child landmarks $\{l_j^k\}$ are marginalized out of the local VO module, we freeze the landmark inverse depth values ξ_j and insert a new pose $\mathbf{S}_k^W \in \mathrm{Sim}(3)$ into a global pose graph $\mathcal{G}_G = (\mathcal{V}_G, \mathcal{F}_G)$. Here $\mathcal{V}_G$ contains only $\mathrm{Sim}(3)$ pose variables. The factor set $\mathcal{F}_G$ contains relative odometry factors, loop closure factors, and scale factors.

1) Relative Odometry Factors: We link the newly inserted pose variable k to the rest of $\mathcal{V}_G$ using a relative odometry factor r_{odom} between k and the most recent global pose variable j. Let $\hat{\mathbf{S}}_k^j$ denote the relative transform between k and j when k is marginalized out of the local window. r_{odom} is then given by

$$r_{odom}(\mathbf{S}_k^W, \mathbf{S}_j^W) = \log\left(\mathbf{S}_W^j\mathbf{S}_k^W\hat{\mathbf{S}}_j^k\right), \tag{11}$$

where $\log : \mathrm{Sim}(3) \to \mathfrak{sim}(3)$ denotes the logarithmic map between $\mathrm{Sim}(3)$ and its Lie algebra $\mathfrak{sim}(3)$. We let $\mathcal{F}_{odom}$ denote the set of all odometry factors.

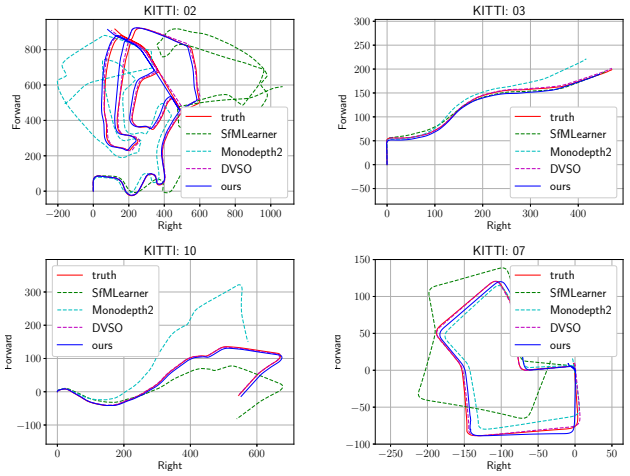

Fig. 4: *KITTI Odometry Performance*: Our method gives compelling performance on the KITTI odometry benchmark. The left column shows sequences that were included in the training data of our metric depth prediction network. The right column shows sequences that were not used to train the network.

2) Loop Closure Factors: When local keyframe k is marginalized out, we compare I_k to the images corresponding to the poses in $\mathcal{V}_G$ using a bag-of-words (BoW)-based descriptor vector [35] generated with ORB features [36]. If a match is detected, we then match the features across the two frames and use the matches to estimate the relative Sim(3) transform between the two poses. A loop factor r_{loop} is then added to $\mathcal{F}_G$ with the same form as r_{odom}. We let $\mathcal{F}_{loop}$ denote the set of all loop factors.

3) Scale Factors: We employ the inverse depth estimation network described in Section III-B to generate scale measurements for a pose in the global graph $\mathcal{G}_G$. The child landmarks l_j^i of a pose $\mathbf{S}_i^W$ have arbitrarily scaled inverse depths ξ_j. Using our inverse depth estimation network, we can estimate the metric inverse depth z_j for each landmark. The ratio of the unscaled inverse depth ξ_j to the metric inverse depth z_j is an estimate of the scale s_i of the pose $\mathbf{S}_i^W$. We can add these measurements as unary factors r_s on the scale variable s_i:

$$r_s(\mathbf{S}_i^W) = s_i - \xi_j / z_j. \tag{12}$$

We let $\mathcal{F}_s$ denote the set of all scale factors.

4) Backend: The total cost represented by $\mathcal{G}_G$ can then be written as:

$$\begin{aligned} E_G(\mathcal{V}_G) = &\sum_{j,k \in \mathcal{F}_{odom}} \left|\left| r_{odom}(\mathbf{S}_k^W, \mathbf{S}_j^W) \right|\right|^2_{\Sigma_{odom}} + \\ &\sum_{j,k \in \mathcal{F}_{loop}} \left|\left| r_{loop}(\mathbf{S}_k^W, \mathbf{S}_j^W) \right|\right|^2_{\Sigma_{loop}} + \\ &\sum_{i \in \mathcal{F}_s} ||r_s(\mathbf{S}_i^W)||^2_{\epsilon_s} \end{aligned} \tag{13}$$

and can be optimized using Levenberg-Marquardt [33]. Here $\Sigma_{odom}, \Sigma_{loop} \in \mathbb{R}^{7\times7}$ denote the odometry and loop noise covariances, respectively and ϵ_s denotes the Huber noise parameter for the scale factors.

KITTI Odometry Benchmark

		SfMLearner [18]		Monodepth2 [20]		DVSO [26]		Ours	
	Run	t_{rel}	r_{rel}	t_{rel}	r_{rel}	t_{rel}	r_{rel}	t_{rel}	r_{rel}
Train	02	11.0	4.18	13.1	5.27	0.84	0.22	1.15	0.27
	06	10.7	6.31	17.0	12.9	0.73	0.35	2.61	1.22
	08	8.93	3.75	14.2	5.98	1.03	0.25	1.71	0.35
	09	10.6	4.07	17.7	6.18	0.83	0.21	1.70	0.48
	10	11.1	4.06	13.1	6.74	0.74	0.21	1.01	0.37
	Avg	10.4	4.11	13.8	5.56	0.89	0.23	1.39	0.33
Test	00	15.9	6.19	15.5	6.47	0.71	0.24	4.55	0.93
	03	11.1	4.52	10.2	2.93	0.77	0.18	4.72	0.21
	04	3.69	3.28	10.6	1.46	0.35	0.06	18.8	0.27
	05	10.8	4.66	12.6	6.51	0.58	0.22	2.36	0.33
	07	12.7	5.58	10.1	3.25	0.72	0.20	1.09	0.30
	Avg	13.7	5.63	14.4	6.69	0.67	0.24	3.85	0.73

TABLE I: *KITTI Odometry Benchmark*: Here we show our pipeline's performance on the KITTI Odometry Benchmark [1]. t_{rel} denotes the relative translation error averaged over 100m to 800m path segments (expressed as a percent of distance traveled). r_{rel} denotes the relative rotation error averaged over the same path segments (expressed as degrees per 100m). The runs labeled "Train" are included in the training data for both our network and DVSO [26], while the runs labeled "Test" are not. Note that our method performs competitively on the benchmark despite only requiring a CPU.

IV. Evaluation

We demonstrate the performance of our approach quantitatively using the KITTI Odometry Benchmark [1] (Section IV-B) and qualitatively using handheld imagery collected from an indoor environment (Section IV-C).

A. Implementation Details

We designed our depth prediction network using Tensorflow [37] and set the base image size $L0$ to 256×512 pixels. For all our experiments, the network is trained for 100 epochs using the Adam optimizer [38] on an NVIDIA 1080Ti GPU with a batch size of 8 and a learning rate of 0.0001, which is halved after 30 epochs and again after 40 epochs. We follow standard data augmentation practices by randomly flipping the training images left to right and perturbing the image color, including gamma and brightness shifting. We set the weights governing the terms in the loss function as $\lambda_p = 1.0$, $\lambda_{lr} = 1.0$, $\lambda_r = 0.1$, and $\lambda_s = 10.0$. Network inference is triggered at runtime using the REST API of the `tensorflow_serving` package.

Our geometric SLAM pipeline is implemented in C++ using the Ceres solver library [39]. Disparity maps from $L3$ (32×64 pixels) are used generate the metric scale factors for each keyframe pose. Both network inference and SLAM optimization are performed at runtime entirely on an Intel i7 4820K CPU.

B. KITTI Odometry Evaluation

We evaluate the odometry performance of our approach quantitatively using ten video sequences from the KITTI Odometry Benchmark [1]. We train our depth prediction network using the common training split of the raw KITTI stereo data from [40], which consists of 22,600 training stereo pairs, 888 validation pairs, and 697 testing pairs.

Of the ten odometry sequences, images from runs 00, 06, 08, 09, and 10 are included in the training data of the

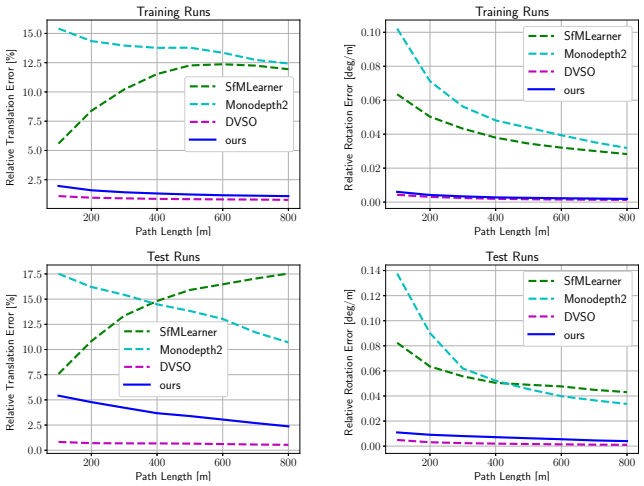

Fig. 5: *Relative Pose Error vs. Distance Traveled*: The plots above show the relative translation (left) and rotation error (right) on the training (top) and test (bottom) runs from the KITTI Odometry Benchmark [1]. Our method achieves competitive performance on the benchmark despite not relying on GPU acceleration.

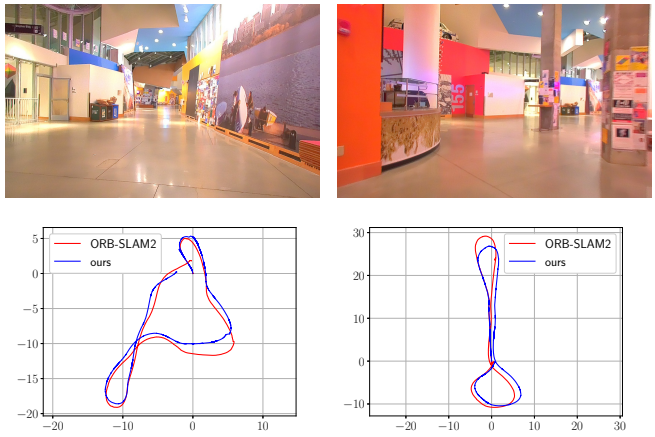

Fig. 6: *Handheld Trajectories*: We demonstrate our method's performance using handheld camera data from an indoor environment. *Top row*: Sample training images from the environment. *Bottom row*: Comparison of poses from our approach (blue) against those of Stereo ORB-SLAM2 [42] on two test trajectories. Note that Stereo ORB-SLAM2's poses are correctly scaled as the stereo baseline is known *a priori*. Our technique is able to generate correctly scaled poses using only a single monocular camera.

depth prediction network. Runs 00, 03, 04, 05, and 06 have no overlap with the depth network training data. (Run 01 exhibits very little texture for feature detection and was not used to evaluate odometry.) Example trajectories for training and test runs are shown in Figure 4.

Quantitative performance is measured using relative pose error (RPE) [41] over a set of predefined path lengths (100 m to 800 m). Table I shows the relative translation error t_{rel} (expressed as a percent of distance traveled) and relative rotation error r_{rel} (expressed in degrees per 100m) for each run averaged over all path lengths, while Figure 5 shows these metrics for each path length averaged over all runs.

We compare our method against two end-to-end SLAM packages (SfMLearner [18] and Monodepth2 [20]) that are scaled to metric scale and a hybrid learning/geometric approach DVSO [26]. Note that SfMLearner and Monodepth2 are trained on runs 00-08. DVSO is trained using the same split as our method, but uses additional supervision from a sparse reconstruction method [7]. We are unable to do a full comparison to DVSO as the authors have not provided a public implementation of their technique, and so we compare to their published results.

Our method performs competitively on the benchmark, achieving a relative translation error of 1.39 percent and a relative rotation error of 0.33° / 100m on the training runs and 3.85 percent and 0.73° / 100m on the test runs. Notably, all computation is performed entirely on the CPU, while other methods require GPU acceleration. Network inference takes approximately 30 ms per frame. For comparison, the authors of DVSO report that evaluations of their network take 40 ms per frame on an NVIDIA Titan X Pascal GPU.

C. Handheld Odometry Evaluation

In addition to the quantitative results described in the previous section, we also qualitatively validate our system with imagery collected using a handheld stereo camera that captures time synchronized images at 16 Hz. The baseline between the left and right cameras is 5 cm. The environment used for the experiment is a large, indoor laboratory common area and student thoroughfare between classrooms.

We collected a total of 16,980 stereo images, 11,548 of which were used for training our depth prediction network with 1,510 pairs used for validation. Two complete runs comprising 3,922 pairs were withheld to test our odometry performance. At runtime, the images from the left camera were used to compute our metrically scaled poses. The trajectories for the two test runs are shown in Figure 6. In the absence of groundtruth poses, we compare our monocular odometry estimates against that of Stereo ORB-SLAM2 [42], a state-of-the-art geometric stereo odometry pipeline. (Note that as a stereo method, its poses are metrically scaled since the baseline between the left and right cameras is known.) As evident in Figure 6, our method is able to produce accurate poses at the correct metric scale despite only using a single monocular camera.

V. Conclusion

We have proposed an efficient method for monocular SLAM that is capable of estimating metrically-scaled motion without additional sensors or compute by integrating metric depth predictions from a neural network into a geometric SLAM pipeline. Our depth prediction network is designed specifically for metric scale estimation and thus can be much smaller and faster than competing systems. We make several improvements to our network architecture and training procedure to address the lack of depth observability when using coarse image input that allows us to estimate spatially coarse, but depth-accurate predictions in only 30 ms per frame. We show compelling results on the KITTI benchmark dataset in addition to real-world experiments with a handheld camera.

References

[1] A. Geiger, P. Lenz, and R. Urtasun, "Are we ready for Autonomous Driving? The KITTI Vision Benchmark Suite," in *Proc. CVPR*, 2012.

[2] A. J. Davison, "Real-time simultaneous localisation and mapping with a single camera," in *Proc. ICCV*, 2003.

[3] G. Klein and D. Murray, "Parallel tracking and mapping for small AR workspaces," in *Proc. ISMAR*, 2007.

[4] R. Mur-Artal, J. Montiel, and J. D. Tardos, "ORB-SLAM: A versatile and accurate monocular slam system," *Trans. on Robotics*, 2015.

[5] C. Forster, M. Pizzoli, and D. Scaramuzza, "SVO: Fast semi-direct monocular visual odometry," in *Proc. ICRA*, 2014.

[6] J. Engel, T. Schöps, and D. Cremers, "LSD-SLAM: Large-scale direct monocular slam," *Proc. ECCV*, 2014.

[7] J. Engel, V. Koltun, and D. Cremers, "Direct sparse odometry," *arXiv:1607.02565*, 2016.

[8] D. Gálvez-López, M. Salas, J. D. Tardós, and J. Montiel, "Real-time monocular object SLAM," *Robotics and Autonomous Systems*, 2016.

[9] A. I. Mourikis and S. I. Roumeliotis, "A multi-state constraint Kalman filter for vision-aided inertial navigation," in *Proc. ICRA*, 2007.

[10] S. Leutenegger, S. Lynen, M. Bosse, R. Siegwart, and P. Furgale, "Keyframe-based visual–inertial odometry using nonlinear optimization," *IJRR*, 2015.

[11] C. Forster, L. Carlone, F. Dellaert, and D. Scaramuzza, "IMU Preintegration on Manifold for Efficient Visual-Inertial Maximum a Posteriori Estimation," in *RSS*, 2015.

[12] T. J. Steiner, R. D. Truax, and K. Frey, "A vision-aided inertial navigation system for agile high-speed flight in unmapped environments," in *Proce. IEEE Aerospace Conference*, 2017.

[13] T. Qin, P. Li, and S. Shen, "VINS-Mono: A robust and versatile monocular visual-inertial state estimator," *arXiv:1708.03852*, 2017.

[14] S. Wang, R. Clark, H. Wen, and N. Trigoni, "DeepVO: Towards end-to-end visual odometry with deep recurrent convolutional neural networks," in *Proc. ICRA*, 2017.

[15] R. Li, S. Wang, Z. Long, and D. Gu, "UnDeepVO: Monocular visual odometry through unsupervised deep learning," *arXiv:1709.06841*, 2017.

[16] R. Clark, S. Wang, H. Wen, A. Markham, and N. Trigoni, "VINet: Visual-inertial odometry as a sequence-to-sequence learning problem.," in *AAAI*, 2017.

[17] G. Iyer, J. K. Murthy, G. Gupta, K. M. Krishna, and L. Paull, "Geometric consistency for self-supervised end-to-end visual odometry," *arXiv:1804.03789*, 2018.

[18] T. Zhou, M. Brown, N. Snavely, and D. G. Lowe, "Unsupervised learning of depth and ego-motion from video," in *Proc. CVPR*, 2017.

[19] C. Godard, O. Mac Aodha, and G. J. Brostow, "Unsupervised monocular depth estimation with left-right consistency," in *Proc. CVPR*, 2017.

[20] C. Godard, O. Mac Aodha, M. Firman, and G. J. Brostow, "Digging into self-supervised monocular depth prediction," *arXiv:1806.01260*, 2018.

[21] M. Poggi, F. Aleotti, F. Tosi, and S. Mattoccia, "Towards real-time unsupervised monocular depth estimation on CPU," in *Proc. IROS*, 2018.

[22] A. Saxena, M. Sun, and A. Y. Ng, "Make3D: Learning 3D scene structure from a single still image," *Trans. PAMI*, 2008.

[23] F. Liu, C. Shen, G. Lin, and I. Reid, "Learning depth from single monocular images using deep convolutional neural fields," *Trans. PAMI*, 2015.

[24] K. Tateno, F. Tombari, I. Laina, and N. Navab, "CNN-SLAM: Real-time dense monocular slam with learned depth prediction," in *Proc. CVPR*, 2017.

[25] V. Peretroukhin and J. Kelly, "DPC-Net: Deep pose correction for visual localization," *IEEE Robotics and Automation Letters*, 2018. DOI: 10.1109/LRA.2017.2778765.

[26] N. Yang, R. Wang, J. Stueckler, and D. Cremers, "Deep virtual stereo odometry: Leveraging deep depth prediction for monocular direct sparse odometry," in *Proc. ECCV*, 2018.

[27] R. Wang, M. Schworer, and D. Cremers, "Stereo DSO: Large-scale direct sparse visual odometry with stereo cameras," in *Proc. ICCV*, 2017.

[28] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proc. CVPR*, 2016.

[29] V. Nair and G. E. Hinton, "Rectified linear units improve restricted boltzmann machines," in *Proc. ICML*, 2010.

[30] J. Shi and C. Tomasi, "Good features to track," in *Proc. CVPR*, 1994.

[31] B. D. Lucas, T. Kanade, *et al.*, "An iterative image registration technique with an application to stereo vision.," in *IJCAI*, 1981.

[32] S. Baker and I. Matthews, "Lucas-Kanade 20 years on: A unifying framework," *IJCV*, 2004.

[33] D. W. Marquardt, "An algorithm for least-squares estimation of nonlinear parameters," *Journal of the society for Industrial and Applied Mathematics*, 1963.

[34] J. Nocedal and S. Wright, *Numerical optimization*. Springer Science & Business Media, 2006.

[35] R. Muoz-Salinas and R. Medina-Carnicer, "UcoSLAM: Simultaneous Localization and Mapping by Fusion of KeyPoints and Squared Planar Markers," Feb. 2019. DOI: 10.13140/RG.2.2.31751.65440.

[36] E. Rublee, V. Rabaud, K. Konolige, and G. Bradski, "ORB: An efficient alternative to SIFT or SURF," in *Proc. ICCV*, 2011.

[37] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, A. Davis, J. Dean, M. Devin, S. Ghemawat, I. Goodfellow, A. Harp, G. Irving, M. Isard, Y. Jia, R. Jozefowicz, L. Kaiser, M. Kudlur, J. Levenberg, D. Mané, R. Monga, S. Moore, D. Murray, C. Olah, M. Schuster, J. Shlens, B. Steiner, I. Sutskever, K. Talwar, P. Tucker, V. Vanhoucke, V. Vasudevan, F. Viégas, O. Vinyals, P. Warden, M. Wattenberg, M. Wicke, Y. Yu, and X. Zheng, *TensorFlow: Large-scale machine learning on heterogeneous systems*, Software available from tensorflow.org, 2015. [Online]. Available: `http://tensorflow.org/`.

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

[39] S. Agarwal, K. Mierle, and Others, *Ceres solver*, `http://ceres-solver.org`.

[40] 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*, 2014.

[41] J. Sturm, N. Engelhard, F. Endres, W. Burgard, and D. Cremers, "A benchmark for the evaluation of RGB-D slam systems," in *Proc. IROS*, 2012.

[42] R. Mur-Artal and J. D. Tardós, "ORB-SLAM2: An open-source SLAM System for Monocular, Stereo and RGB-D cameras," *Trans. on Robotics*, 2017.