Received March 26, 2020, accepted May 5, 2020, date of publication May 11, 2020, date of current version May 22, 2020.

Digital Object Identifier 10.1109/ACCESS.2020.2993494

Learning Depth for Scene Reconstruction Using an Encoder-Decoder Model

XIAOHAN TU[1,2], **CHENG XU**[1,2], **SIPING LIU**[1,2], **GUOQI XIE**[1,2], **(Senior Member, IEEE), JING HUANG**[1,2], **RENFA LI**[1,2], **(Senior Member, IEEE), AND JUNSONG YUAN**[3], **(Senior Member, IEEE)**

[1]Key Laboratory for Embedded and Network Computing of Hunan Province, Changsha 410082, China
[2]College of Computer Science and Electronic Engineering, Hunan University, Changsha 410082, China
[3]Department of Computer Science and Engineering, State University of New York at Buffalo, Buffalo, NY 14260, USA

Corresponding author: Cheng Xu (chengxu@hnu.edu.cn)

This work was supported by the National Natural Science Foundation of China under Grant 61772185.

ABSTRACT Depth estimation has received considerable attention and is often applied to visual simultaneous localization and mapping (SLAM) for scene reconstruction. At least to our knowledge, sufficiently reliable depth always fails to be provided for monocular depth estimation-based SLAM because new image features are rarely re-exploited effectively, local features are easily lost, and relative depth relationships among depth pixels are readily ignored in previous depth estimation methods. Based on inaccurate monocular depth estimation, SLAM still faces scale ambiguity problems. To accurately achieve scene reconstruction based on monocular depth estimation, this paper makes three contributions. (1) We design a depth estimation model (DEM), consisting of a precise encoder to re-exploit new features and a decoder to learn local features effectively. (2) We propose a loss function using the depth relationship of pixels to guide the training of DEM. (3) We design a modular SLAM system containing DEM, feature detection, descriptor computation, feature matching, pose prediction, keyframe extraction, loop closure detection, and pose-graph optimization for pixel-level scene reconstruction. Extensive experiments demonstrate that the DEM and DEM-based SLAM are effective. (1) Our DEM predicts more reliable depth than the state of the arts when inputs are RGB images, sparse depth, or the fusion of both on public datasets. (2) The DEM-based SLAM system achieves comparable accuracy as compared with well-known modular SLAM systems.

INDEX TERMS Convolutional neural networks, depth estimation, decoder, encoder, simultaneous localization and mapping.

I. INTRODUCTION

Considered as an important computer vision topic, depth estimation focuses on predicting depth from RGB images (monocular images), sparse depth, or the fusion of both (RGBd) [1]. The predicted depth is often used for various tasks, such as visual simultaneous localization and mapping (SLAM), robot localization, obstacle avoidance, and semantic segmentation [2]. When applying depth prediction to SLAM, monocular camera-based SLAM will be low-cost and attractive as compared with RGBD camera-based SLAM. Essentially, depth estimation creates a virtual RGBD sensor for monocular SLAM, helping SLAM contribute to industries such as robots and self-driving cars.

Recently, most previous work uses convolutional neural networks (CNNs) to predict depth from sparse depth [3], monocular RGB images [1], [4], [5], [6], or the fusion of both [7], [8]. These prior methods tend to leverage CNN models like ResNets [9] to learn visual features, such as the research [5]–[8]. Here, ResNets are often adopted as encoders [5], [8], and these encoders reuse image features in depth estimation. After encoders, decoders including linear interpolation are commonly used to output high-resolution depth maps, yet the decoders easily lose local image features. In essence, recent encoders and decoders suffer from accuracy limitation imposed by their respective shortcomings of effectively re-exploring new features and learning local features in depth prediction. Additionally, researchers often estimate depth with CNNs driven by ground-truth metric depth, usually ignoring the use of relative depth relationships among pixels in RGB images. We find that CNNs are trained better

The associate editor coordinating the review of this manuscript and approving it for publication was Jenny Mahoney.

by loss functions combining relative depth relationship with ground-truth metric depth. The relative depth relationship can help loss functions minimize prediction error and penalize larger outliers in depth estimation. For reliable depth results using the relative depth relationship, we need to redesign a loss function.

Monocular depth estimation often provides depth for SLAM to reconstruct scenes such as studies [10], [11], but inaccurate monocular depth estimation brings more error to SLAM. In addition to this problem, other challenges still exist in depth estimation-based SLAM. Specifically, Tateno *et al.* [10] only refined depth predictions from high-gradient pixels, yet depth predictions of low-gradient pixels also need to be improved to reconstruct reliable scenes. Luo *et al.* [11] needed images with a fixed baseline and just serve horizontal motion of cameras. Tang *et al.* [12] faced multi-frame matching setup problems and had difficulty in practical application. Therefore, to better perform monocular reconstruction without issues such as multi-frame matching, a plug-and-play SLAM should be developed, alleviating scale ambiguity.

This paper proposes a dependable encoder, decoder, loss function, and SLAM system. The encoder is designed with dual path networks (DPNs) [13] to reuse and re-exploit features. Specifically, DPNs inherit advantages of ResNets [9] and DenseNets [14] by respectively adopting their residually and densely connected paths. Through the altered DPN structure, our encoder can re-explore new features flexibly. The decoder is proposed with transposed convolution and convolution layers to recover details lost by linear interpolation in existing decoders. Our decoder can learn dense depth maps accurately.

To further boost depth estimation accuracy, we develop a loss function by using relative relationships among depth points in RGB images, guiding the training of the encoder-decoder model (DEM). The loss function encourages our predicted depth to agree with ground-truth depth. Relying on DEM, monocular SLAM is designed, consisting of eight independent modules: DEM, feature detection, descriptor computation, feature matching, pose prediction, keyframe extraction, loop closure detection, and pose-graph optimization. These modules are easy-to-use and plug-and-play.

Experiments show that the DEM and DEM-based SLAM are more accurate than the state of the arts under the same condition. For example, the RMSE of DEM is 17.4% better than that of the representative work [15] with monocular RGB inputs from the NYU-Depth-v2 dataset.[1] The RMSE of DEM is decreased by 29.3% than that of the classic method [8] with RGB inputs from the KITTI dataset.[2] Our RMSE is 18.1% lower than that of the state of the art [16] with RGBd-500 inputs (each RGB input image contains 500 valid depth samples) on KITTI. As shown in Fig. 1(c), pixel-level depth maps are estimated from RGB images by DEM. The prediction of DEM is identified more easily than that of the study [8] in Fig. 1(b). Based on depth estimated by DEM, our SLAM achieves greater improvement in accuracy on the TUM dataset than others such as SLAM [10]–[12], [17], [18]. Additionally, our SLAM is low-cost as compared with RGBD camera-based SLAM [19], [20]. The results demonstrate that the DEM and DEM-based SLAM are effective.

[1]https://cs.nyu.edu/ silberman/datasets/nyu_depth_v2.html
[2]http://www.cvlibs.net/datasets/kitti

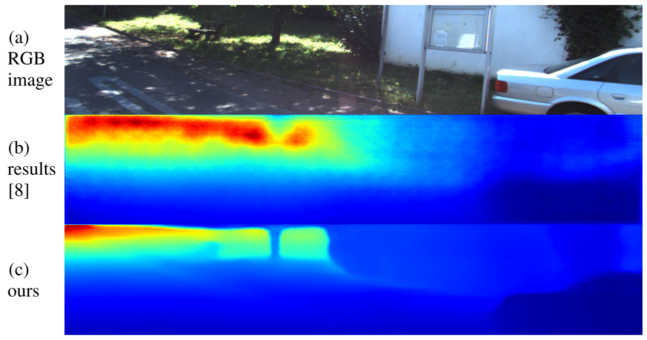


FIGURE 1. **Monocular depth prediction on KITTI. (a) the RGB input; (b) the prediction in the work [8] from the RGB input; (c) our prediction from the RGB input.**

To summarize, our main contributions are as follows.

- First, we propose the DEM architecture containing a reliable encoder and decoder. Specifically, the encoder effectively reuses and re-explores features relying on existing CNNs. The decoder learns local features which are easily lost by linear interpolation in other decoders. Our decoder is capable of combining different encoders and consistently improves depth estimation performance.
- Second, a loss function is designed to guide the training of DEM. Our results verify that the loss function achieves comparable accuracy, whether inputs are RGB images or sparse depth. Additionally, we deploy and evaluate DEM on embedded devices. Here, DEM satisfies real-time (9 frames per second) and low-power (6.9 W) constraints with RGB image inputs of size 480×640 on embedded devices.
- Third, relying on DEM, we develop a low-cost SLAM system including plug-and-play modules for effective scene reconstruction. The DEM-based SLAM estimates real object scales and reconstructs scenes reliably, although using monocular images. Our SLAM achieves more promising results than others on the TUM dataset. We confirm the DEM-based SLAM is useful to researchers who reconstruct scenes with monocular cameras because DEM is precise, monocular cameras employed by our SLAM are generally cheaper than RGBD cameras, and our SLAM modules are easy-to-use.

To our knowledge, the gap between depth prediction and its application in SLAM is reduced by this work.

II. RELATED WORK

In this section, we overview recent studies that are most relevant to our work on two subjects: depth prediction and depth estimation-based SLAM systems.

Previous research on depth estimation can be divided into three categories: RGB-based, multimodal data-based, and sparse depth-based depth estimation. The RGB-based depth estimation often relies on traditional machine learning methods or CNNs. For example, Saxena *et al.* [21], [22] used traditional markov random fields to predict depth. Karsch *et al.* [23] leveraged a non-parametric approach to estimate depth of dynamic foreground objects in video and static backgrounds in a single image. These pioneer studies laid the foundation for RGB-based depth estimation. Eigen *et al.* [4] first used CNNs to infer depth from RGB images. Then, Eigen and Fergus [24] extended two-scale networks to three-scale networks for depth prediction. Roy and Todorovic [25] presented random forests and CNNs to infer depth. Wang *et al.* [26] employed perceptual losses to estimate depth.

Some work [16], [27] focused on fast depth estimation, but these methods still have a scale ambiguity problem because they improve inference speed at the cost of less accuracy. Other research estimated depth with ResNets such as [5], [6], [8], [15], [28], [29], [30]–[33], [34]. In these studies, ResNets effectively reused features through residually connected paths. Different from ResNets, DenseNets [14] efficiently re-explored new features by densely connected paths. Then, inheriting the backbone architecture of ResNets and DenseNets, DPNs [13] simultaneously reused and re-explored features with residually and densely connected paths. In general, effective feature exploration and utilization in DPNs can bring more accuracy to depth estimation, but fewer methods leverage DPNs to predict depth.

Multimodal data-based depth estimation commonly uses inputs containing two or three modalities of data [7], [8], [35]–[37]. The method [7] converted depth estimation into distance prediction between reference and true depth maps, performing more effectively than the depth prediction [35]. Wang *et al.* [36] inferred depth by iteratively changing intermediate representation in pre-trained depth estimation models. Li *et al.* [37] employed depth samples and RGB images to estimate depth. Sparse depth-based depth prediction also used deep learning models [3], [8], [38] to predict depth. Specifically, Chodosh *et al.* [3] employed alternating direction neural networks and compressed sensing techniques to extract features. The research [8], [38] selected CNNs to recover dense depth maps with linear interpolation which easily lost depth features. These problems still need to be addressed for effective depth estimation and depth estimation-based SLAM.

Existing visual SLAM often relies on data from stereo cameras, monocular cameras [10], [11], [17], [18], [39], or RGBD cameras [12], [19], [20]. In research [10]–[12], [20], [39], the SLAM systems all employ deep learning techniques to extract features. Specifically, in approaches [12], [20], keypoints and descriptors are learned by CNN and RNN (recurrent neural network) for SLAM. In methods [10], [11], [39], only CNN was integrated into SLAM to improve scene reconstruction. By using CNN-based depth estimation, the studies [10], [11], [39] focused on issues in conventional triangulation and consecutive view matching. For example, Yang *et al.* [39] leveraged a monocular camera to reconstruct dense maps with generative adversarial networks. Tateno *et al.* [10] used CNN to estimate depth maps, which were adopted as the initial guess of keyframes. Then, they revised depth through triangulation and high-gradient pixel matching, but depth estimation of low-gradient pixels is unrefined in CNN-SLAM [10]. Luo *et al.* [11] fused online-adapted CNN with direct monocular SLAM. The work [11] alleviated scale ambiguity and low map completeness, but it is only applicable to horizontal motion of cameras. To solve these problems and further boost the performance of monocular camera-based SLAM, we propose the depth estimation-based SLAM system for precise scene reconstruction.

III. METHOD

To predict depth maps accurately, we design the depth estimation model (DEM) in Section III-A. In Section III-B, a loss function is proposed to guide the training of DEM. In Section III-C, we present the SLAM system based on DEM.

A. DEM

To improve depth estimation, we develop the DEM architecture containing an encoder and a decoder. Here, the capabilities of DEM are verified by three modalities of data as inputs, namely, RGB images, sparse depth, or RGBd data. The type of inputs is the same for model training and inference. The DEM architecture with RGB inputs is displayed as an example in Fig. 2.

Encoder: The encoder is implemented to extract image features. We investigate classic models, including different layers of CNNs, such as ResNet-18, ResNet-34, ResNet-50, DenseNet-121, DPN-68, DPN-92, and DPN-131. In these CNNs, ResNet-50 is employed as an encoder by famous studies [6], [8], [32], [33]. Although few methods use DPNs to predict depth, we modify DPN-92 as the encoder of DEM after considering the tradeoff between multiply-and-accumulate operations and accuracy of DPNs. The initial DPNs [13] are only proposed based on RGB inputs. To process different modalities of inputs, we alter the structure of DPN-92. Additionally, the original DPNs are usually used for three tasks: semantic segmentation, image classification, and object detection. For depth estimation tasks in this paper, we modify DPN-92 to achieve encoding effectively.

DPN-92 is changed as an encoder in four steps. 1) The first layer in DPN-92 is modified with configuration parameter options. Thus, DPN-92 can learn features from RGB images, sparse depth, or RGBd data. The parameter options change

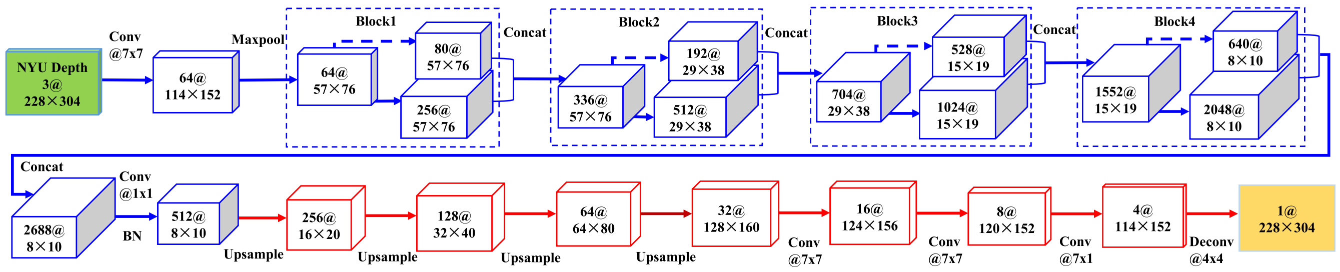


FIGURE 2. The DEM architecture based on RGB image inputs of size 480 × 640. Our DEM consists of an encoder (highlighted in blue) and a decoder (highlighted in red). The training images are augmented to generate RGB images of size 228 × 304. In this figure, a cube represents a feature map. The dimension of each feature map is denoted as #features@height×width. In the encoder, a dotted box means a block in DPN-92, such as block1, block2, block3, and block4. The dotted line in a dotted box represents the shortcut connection between blocks. Concat among dotted boxes means the concat of feature maps. The decoder includes four upsampling layers, two 7 × 7 convolution layers, a 7 × 1 convolution layer, and a 4 × 4 transposed convolution layer. The output of DEM is a pixel-level depth map of size 1@228 × 304.

with the type of inputs. 2) To connect decoders and predict pixel-level depth maps, we remove the last classifier layer in DPN-92. 3) Followed by DPN-92, we use convolution with a kernel size of 1 × 1 and batch normalization (BN) layers. 4) The pre-trained model of DPN-92 on the ImageNet dataset is selected to initialize the encoder of DEM. The initialization improves prediction accuracy. We will verify the benefit of the initialization in Table 4.

Decoder: To output high-resolution and precise depth maps, we propose a decoder, as shown in the red part of Fig. 2. The decoder is designed in the following steps. 1) To upsample feature maps from encoders, we use four upsampling layers consisting of transposed convolutions. 2) To further extract features, three convolution layers are employed. These convolution layers also adjust the size of depth images to produce feature maps of size 114 × 152. 3) To generate pixel-wise depth images of size 228 × 304, a transposed convolution layer is leveraged. In dense depth prediction, our decoder is superior because the pixel-level depth maps are learned by transposed convolution and convolution layers. By contrast, the state of the arts [5], [8], [32] use decoders including linear interpolation.

In the decoder where linear interpolation is used, local image features are easily lost and image features are rarely learned and extracted. These two disadvantages are caused by curve fitting using linear polynomials in the linear interpolation. For example, a simple linear interpolation method is as follows: obtaining the depth value x on the straight line that is determined by the coordinates of two known depth points (x_0, y_0) and (x_1, y_1), where the value x is in the interval $[x_0, x_1]$. Similarly, bilinear interpolation adopted in existing decoders is essentially linear interpolation in two directions. In linear interpolation methods, we can see that the depth value x is not obtained by learning or leveraging local features. In contrast, our decoder addresses the above issue of losing local features easily and extracting no features. The following experiments in Table 2 will demonstrate the effectiveness of our decoder.

B. LOSS FUNCTION

To predict precise depth, we propose the loss function L guiding the training of DEM. In general, it is challenging to estimate depth with a monocular image due to scale ambiguities. Although depth of objects in an image is ambiguous, the depth between different points has a relative relationship. The relationship of depth points is easy to access. Therefore, we use the relationship among sparse depth samples to design the loss function L as follows:

$$L = \mathcal{L}_1 + R, \tag{1}$$

where $\mathcal{L}_1$ is

$$\mathcal{L}_1 = \frac{1}{N}\sum_{s=1}^{N} |P_s - T_s|, \tag{2}$$

and N denotes the number of pixels in a depth map; P_s and T_s ($s = 1, 2, 3, \ldots, N$) represent depth values of pixels in predicted and ground-truth maps, respectively.

The second part R is based on relationships among depth samples as follows:

$$R = \frac{1}{M}\sum_{m=1}^{M} \left|\left(T_{i_m} - T_{j_m}\right) - \left(P_{i_m} - P_{j_m}\right)\right|. \tag{3}$$

Specifically, we randomly combine two depth points in the set D consisting of sparse depth samples in a depth image. Then, we obtain a combination of depth points from the set D. The number of the combination is represented as M. The pair in the combination is denoted as $\{(i_m, j_m)\}$, where $m = 1, 2, 3, \ldots, M$. The locations of the first and second point of a pair in the M-th combination are i_m and j_m, respectively; P_{i_m} and P_{j_m} respectively represent depth values at points i_m and j_m in a predicted depth map; T_{i_m} and T_{j_m} respectively denote depth values at points i_m and j_m in a ground-truth depth map; $\left(T_{i_m} - T_{j_m}\right)$ and $\left(P_{i_m} - P_{j_m}\right)$ represent the relationship of depth between i_m and j_m in ground-truth and predicted depth maps, respectively.

The second part R has three advantages. 1) R enhances the effect of $\mathcal{L}_1$. Generally, R minimizes prediction error. 2) R pushes predicted depth further closer to true depth. 3) R assigns a large penalty for large outliers and still penalizes small outliers.

By combining R and $\mathcal{L}_1$, we obtain the loss function L. Experimental results will demonstrate that L outperforms other loss functions. In essence, L performs well by taking advantage of $\mathcal{L}_1$ and R. Specifically, the loss function L has two benefits as follows.

First, our loss function L encourages estimated depth to agree with the ground truth. The first part of L is effective. The second part of L fully uses relative relationships among different sparse depth points. The acquired relationships are about uniform because we obtain sparse depth points randomly and automatically. Through the use of relative relationships, L is robust and reduces the impact of outliers.

Second, our loss function L exceeds other loss functions, such as $\mathcal{L}_2$ and berHu. As a common loss function for regression problems, $\mathcal{L}_2$ may adjust models based on outliers which cause poor depth estimation in previous research. Oppositely, our loss function L is reliable. To demonstrate the effectiveness of our loss function L, we present an empirical study. The results are shown in Table 3.

C. SLAM SYSTEM BASED ON DEM

Our feature-based SLAM alleviates inherent ambiguity problems in monocular camera-based SLAM. As shown in Fig. 3, each box denotes an independent module in SLAM. The details are as follows. Single RGB images are captured by a monocular camera. The RGB images are used by DEM (pre-trained with RGB images) to predict depth maps. By DEM, the real scales of objects are obtained. The keypoints in the RGB images are detected by the ORB algorithm in OpenCV.[3] Based on the detected keypoints, the BRIEF descriptors [40] are computed. Through the modules of feature detection and descriptor computation, we obtain feature points. In the feature matching module, the BRIEF descriptors are matched between two frames of RGB images. Then, we find the minimum distance of the BRIEF descriptors. If the distance of the BRIEF descriptors is less than six times the minimum distance, the matches are considered as good matches. Relying on the good matches of feature points, we obtain the feature points' 3D (three-dimensional) positions by using depth maps estimated by DEM. The 3D positions and pixel coordinates of feature points are as inputs of a pose prediction module.

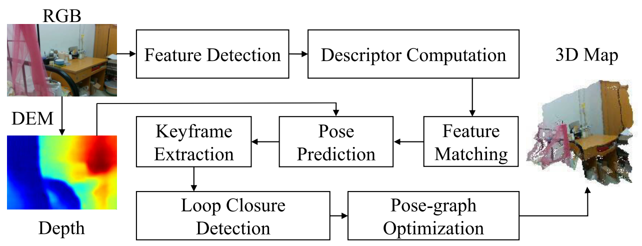


FIGURE 3. The SLAM System.

The pose prediction module is performed as shown in Fig. 3. First, poses of monocular images are estimated by the Random sample consensus Perspective-n-Point (RANSAC PnP) algorithm in OpenCV. Second, we constitute a nonlinear least-squares problem. To optimize the nonlinear

[3]https://opencv.org

Algorithm 1 Pose Prediction

```
Input:
    The pixel and 3D coordinates of feature points in good matches.
Output:
    The frame f2's optimized pose $T$ defined by G2o and the number of inliers $h$.
1: Relying on the 3D and pixel coordinates of feature points, obtain poses using the RANSAC PnP algorithm in OpenCV, and then acquire the 3D rotation vector $r$, the 3D translation vector $t$, and the number of inliers $h$;
2: Create a class "EdgeProjectXYZ2UVPose" relying on Equation (6) to optimize only poses;
3: Create a new object pointer of a linear solver and initialize it;
4: Create a new object pointer of a block matrix solver and initialize it;
5: Select the Levenberg-Marquardt algorithm [42] as the gradient descent method in G2o and sparse optimizer "poseOptimizer";
6: Create a new object pointer of the vertex "vertexPose" belonging to the class "VertexSE3Expmap";
7: Initialize the vertex and set the vertex index to 0;
8: Obtain the pose defined by G2o using $r$ and $t$ in Step 1 of Algorithm 1;
9: Set the pose defined by G2o to the initial value of G2o optimization;
10: Add the vertex to the optimizer "poseOptimizer";
11: for $i \leftarrow 0$ to $h$ do
12:     Create a new object pointer of an edge belonging to the class "EdgeProjectXYZ2UVPose";
13:     Initialize the edge and set the edge index to the number $i$;
14:     Initialize the 3D point and camera parameter variables related to the edge;
15:     Set the observation value of the edge to the 2D point coordinates corresponding to the inlier $i$;
16:     Define a $2 \times 2$ information matrix and use a $2 \times 2$ unit matrix to initialize it;
17:     Add the edge to the optimizer "poseOptimizer";
18: end for
19: Start optimization for 10 iterations and obtain the frame f2's optimized pose $T$ defined by G2o;
20: return $T$ and $h$.
```

least-squares problem, we use G2o [41] to minimize two-dimensional (2D) reprojection errors. The minimization of 2D reprojection errors can generate poses of monocular images. To help accurate pose estimation of monocular images, the results of the RANSAC PnP algorithm are employed as the initial values of the pose optimization in G2o, as shown in Algorithm 1. The initial values in G2o boost the optimization quality of poses. We will describe how to constitute least-squares problems using 2D reprojection errors and optimize the nonlinear least-squares problem by minimizing 2D reprojection errors.

Following the method [19], we constitute the nonlinear least-squares problem. Suppose that 3D positions of n feature points in good matches are denoted as $\boldsymbol{F}_i = [X_i, Y_i, Z_i]^{\mathrm{T}}$ and $i = 1, 2, 3, \ldots, n$. The feature points' pixel coordinates are set to $\boldsymbol{g}_i = [s_i, t_i]^{\mathrm{T}}$ and $i = 1, 2, 3, \ldots, n$. The Lie algebra of an image pose is defined as ξ. The scale factor and camera intrinsics are l_i and $\boldsymbol{K}$. Then, the relationship between the feature point's 3D position $\boldsymbol{F}_i$ and the feature point's pixel coordinate $\boldsymbol{g}_i$ is given as

$$l_i\,[s_i, t_i, 1]^T = \boldsymbol{K} \exp\left(\xi^\wedge\right) [X_i, Y_i, Z_i, 1]^T\,. \tag{4}$$

The homogeneous coordinate of the feature point's 3D position $\boldsymbol{F}_i$ is represented as $\boldsymbol{F}_i = [X_i, Y_i, Z_i, 1]^{\mathrm{T}}$. The homogeneous coordinate of the feature point's pixel coordinate $\boldsymbol{g}_i$ is denoted as $\boldsymbol{g}_i = [s_i, t_i, 1]^{\mathrm{T}}$. Following Equation (4), we obtain

$$l_i\boldsymbol{g_i} = \boldsymbol{K} \exp\left(\xi^\wedge\right) \boldsymbol{F}_i. \tag{5}$$

The 2D reprojection error occurs on the left and right sides of Equation (5) because poses of monocular images are unknown, and noise exists among different image observation points. We represent the process of summing the 2D reprojection errors, constituting a least-squares problem, and finding the relatively optimal pose which minimizes the 2D reprojection errors as follows:

$$\xi^* = \arg\min_{\xi} \frac{1}{2} \sum_{i=1}^{n} \left\| \boldsymbol{g_i} - \frac{1}{l_i} \boldsymbol{K} \exp\left(\xi^\wedge\right) \boldsymbol{F_i} \right\|_2^2 . \tag{6}$$

Through the minimization of 2D reprojection errors, the least-squares problem is optimized, as presented in Algorithm 1. Here, G2o [41] minimizes the 2D reprojection error by setting vertexes and edges. The Levenberg-Marquardt algorithm [42] is selected as the gradient descent method in G2o. We perform G2o optimization for 10 iterations. When completing G2o optimization, we acquire reliable poses between frames.

The modules of feature detection, descriptor computation, feature matching, and pose prediction make up the front-end of our SLAM, namely, visual odometry (VO). As part of our SLAM, the VO estimates the poses of monocular images through G2o optimization, but the VO only focuses on local consistency of camera trajectories. To achieve a globally consistent scene reconstruction, a complete SLAM system is needed as shown in Fig. 3.

Our SLAM system is developed containing VO, keyframe extraction, loop closure detection, and pose-graph optimization. 1) The keyframe extraction selects keyframes to rebuild point cloud maps of scenes. It is unnecessary that a map is built relying on every frame because the relative motion distance between each frame is small. 2) The loop closure detection identifies drift in SLAM by realizing that a previous area in a map is re-visited or not. 3) The pose-graph optimization minimizes drift and optimizes keyframe poses for consistent maps. By using the above three modules, global consistency is guaranteed in our SLAM.

Algorithm 2 Keyframe Extraction

```
Input:
    A frame sequence $\mathbf{A}$ with $n$ frames.
Output:
    The keyframe sequence $\mathbf{B}$ with $o$ keyframes.
 1: $o \leftarrow 0$;
 2: for $ri \leftarrow 1$ to $n$ do
 3:     Obtain the pose $T$ and the number of inliers $h$ between
        the current frame $r_i$ and its previous frame in the
        keyframe sequence $\mathbf{A}$ using Algorithm 1 based on
        Equation (6);
 4:     Calculate the 3D rotation vector $r$ and translation vec-
        tor $t$ relying on the pose $T$;
 5:     Calculate the relative motion distance $dis$ between the
        current frame $r_i$ and its previous frame using $r$ and $t$
        by Equation (7);
 6:     if ($In < 8$ or $dis < 0.1$ or $dis > 0.21$) then
 7:         continue; //Discard the current frame $r_i$;
 8:     end if
 9:     $o \leftarrow o + 1$; // Add the current frame $r_i$ to the keyframe
        sequence $\mathbf{B}$;
10: end for
11: return The keyframe sequence $\mathbf{B}$ with $o$ keyframes.
```

To this end, the main difference between our SLAM and VO is threefold. 1) The keyframe extraction, loop closure detection, and pose-graph optimization are included in our SLAM, while VO does not include these. 2) The SLAM acquires a globally consistent estimate of a scene map, while VO focuses on local consistency. 3) The SLAM optimizes poses, reduces drift, and achieves global consistency without any prior information when reconstructing point cloud maps of scenes. By contrast, the VO predicts poses incrementally and does not solve a drift problem. We will describe the modules of keyframe extraction, loop closure detection, and pose-graph optimization in detail.

As shown in Algorithm 2, in the keyframe extraction module, we extract keyframes and obtain a keyframe sequence $\mathbf{B}$. Here, the frame sequence consists of n frames. The frame used in our SLAM contains a frame index r_i, and $i = 1, 2, 3, \ldots, n$, an RGB map, a depth map estimated by DEM, keypoints extracted by the feature detection module, and descriptors of keypoints corresponding to the frame. Similarly, the keyframes used by our SLAM have the same structure as the frames. The first frame in the frame sequence $\mathbf{A}$ is put into the keyframe sequence $\mathbf{B}$.

In Algorithm 2, when the number of inliers and relative motion distance between the current frame and its previous frame in $\mathbf{A}$ are suited, the keyframe will be added to the keyframe sequence $\mathbf{B}$. The number of inliers can be obtained from Algorithm 1. The threshold on the number of inliers is set to 8 based on our experience. The relative motion distance is computed by the equation

$$dis = \boldsymbol{a} \times |\min(E_r, 2\pi - E_r)| + \boldsymbol{b} \times |E_t|, \tag{7}$$

where E_r represents the L2-norm of the rotation vector r, and E_t denotes the L2-norm of the translation vector t. Here, $\boldsymbol{a}$ and $\boldsymbol{b}$ denote weighting factors. We define $\boldsymbol{a} = 1/3$ and $\boldsymbol{b} = 2/3$ in the keyframe extraction module.

The number of inliers and Equation (7) help us decide if a frame is a keyframe. Specifically, according to Equation (7), we judge whether the relative motion distance between two frames is within a certain range. According to the number of inliers, we determine whether there is enough matching accuracy between adjacent frames.

The loop closure detection is performed to detect accumulated scale drift, as shown in Algorithm 3. Specifically, the keyframe extraction results are used as inputs of Algorithm 3. If a keyframe in the keyframe sequence **B** closely matches six random keyframes in **C** and seven last keyframes in **C**, the keyframe in **B** will be put into the keyframe sequence **C**. The matches are selected or not according to the number of inliers and Equation (7). In Equation (7), we define $\boldsymbol{a} = 1/3$ and $\boldsymbol{b} = 2/3$ for the loop closure detection module. The threshold on the number of inliers is set to seven based on our experience. When the process in Algorithm 3 is completed, we obtain the more accurate keyframe sequence **C** than **B**.

By using the keyframe sequence **C** as inputs, we perform pose-graph optimization with G2o. The detailed G2o optimization is presented in Algorithm 4. Based on Algorithm 4, the pose-graph optimization module optimizes a pose graph with loop closure constraints, as shown in Algorithm 5. Specifically, if a keyframe in the keyframe sequence **C** accurately matches six random keyframes in **D** and seven last keyframes in **D**, the keyframe in **C** will be put into the keyframe sequence **D**. The matches are detected based on the number of inliers and Equation (7). In Equation (7), we fix $\boldsymbol{a} = 2/3$ and $\boldsymbol{b} = 1/3$ for the pose-graph optimization module relying on our experience. The threshold on the number of inliers is set to seven. The method to detect matched keyframes is the same as that in Algorithm 3. The matched keyframe is optimized by G2o for 80 iterations. Through the G2o optimization, loops are re-detected and corrected. After the pose-graph optimization module, we obtain the keyframe sequence **D** including optimized poses.

To reconstruct dense scenes, we backproject depth maps predicted by DEM from the optimized poses in the keyframe sequence **D**. Relying on optimized poses and accurate depth maps outputted by DEM, the point cloud map of an unknown environment is built with monocular images. The reconstruction results can be acquired without indoor or outdoor constraints through the DEM-based SLAM.

The DEM-based SLAM has three advantages of low manufacturing cost, high accuracy, and easy use. 1) The SLAM is low-cost as compared with RGBD camera-based SLAM such as [19], [20]. This is because depth is acquired from DEM with monocular cameras in our SLAM. In contrast, RGBD camera-based SLAM uses RGBD cameras to obtain depth. The RGBD cameras are generally more expensive than monocular cameras. 2) Although using a cheap monocular camera in our SLAM, we alleviate absolute scale ambiguities. The initialization trouble of monocular cameras is also avoided by our SLAM. As compared with other monocular camera-based SLAM, the reconstruction accuracy is greatly improved by the DEM-based SLAM. 3) Our monocular camera-based SLAM is easy-to-use, requiring no additional multi-frame matching setup and indoor/outdoor limits. As described earlier, each module in our SLAM is

Algorithm 3 Loop Closure Detection

```
Input:
    The keyframe sequence B with $o$ keyframes.
Output:
    The new keyframe sequence C with $m$ keyframes.
 1: Create the new keyframe sequence C with $m$ keyframes
    and initialize $m$ to 0;
 2: for $ri \leftarrow 0$ to $o$ do
 3:   if ($m < 7$) then
 4:     for $i \leftarrow 0$ to $m$ do
 5:       Obtain poses $T$ and the number of inliers $h$
          between the frame $r_i$ in keyframe sequences B
          and frame $i$ in keyframe sequences C using Algo-
          rithm 1, and calculate the 3D rotation vector $r$ and
          translation vector $t$ based on the pose $T$, and then
          use $r$ and $t$ to calculate $dis$ based on Equation (7);
 6:       if ($In < 7$ or $dis > 2.1$ or $dis < 0.1$) then
 7:         continue;
 8:       end if
 9:     end for
10:   else
11:     for $i \leftarrow m - 7$ to $m$ do
12:       Perform Step 5 to Step 8 of Algorithm 3;
13:     end for
14:   end if
15:   if ($m < 6$) then
16:     for $i \leftarrow 0$ to $m$ do
17:       Perform Step 5 to Step 8 of Algorithm 3;
18:     end for
19:   else
20:     for $i \leftarrow 0$ to 6 do
21:       Randomly extract the frame $i$ in the keyframe
          sequence C and perform Step 5 to Step 8 of
          Algorithm 3;
22:     end for
23:   end if
24:   $m \leftarrow m + 1$; // Add the frame $r_i$ to the keyframe
      sequence C;
25: end for
26: return The keyframe sequence C with $m$ keyframes.
```

Algorithm 4 G2o Optimization

```
Input:
    Two frames f1 and f2.
1:  Create new object pointers of a linear solver and a block matrix solver and initialize them;
2:  Select the Levenberg-Marquardt algorithm [42] in G2o as the gradient descent method;
3:  Create a sparse optimizer "optimizer";
4:  Create a new object pointer of a vertex belonging to the class "VertexSE3";
5:  Initialize the vertex and define the vertex number as the index of frame;
6:  Set the initial value of the optimization to the unit matrix;
7:  Fix the first vertex and add the vertex to the optimizer "optimizer";
8:  Create a new object pointer of a vertex belonging to the class "VertexSE3";
9:  Initialize the vertex;
10: Define the vertex number as the index of frame f2;
11: Set $T$ to the initial value of the optimization;
12: Add the vertex to the optimizer "optimizer";
13: Create a new object pointer of an edge and initialize it;
14: Set the first and second vertexes connecting the edge to the indexes of frames f1 and f2;
15: Set the robust kernel function to the Huber loss function in G2o;
16: Define the $6 \times 6$ information matrix and set the diagonal value of the information matrix to 100;
17: Set the observation of the edge to the transformation matrix obtained from the output $T$ of Algorithm 1;
18: Add the edge to the optimizer "optimizer";
19: Start optimization for 80 iterations;
20: return $NONE$.
```

Algorithm 5 Pose-Graph Optimization Using Algorithm 4

```
Input:
    The keyframe sequence C with $m$ keyframes.
Output:
    Optimized poses in the keyframe sequence D.
1:  Create the keyframe sequence D with $s$ ($s$=0) keyframes and perform Step 1 to Step 7 of Algorithm 4;
2:  for $ri \leftarrow 0$ to $m$ do
3:      Obtain optimized poses $T$ and the number of inliers $h$ between the current frame $r_i$ and its previous frame in the keyframe sequence C by using Algorithm 1, and set vertexes and edges in G2o by performing Step 8 to Step 18 of Algorithm 4;
4:      if ($s < 7$) then
5:          for $i \leftarrow 0$ to $s$ do
6:              Perform Step 5 to Step 8 of Algorithm 3 and set edges in G2o by performing Step 13 to Step 18 of Algorithm 4;
7:          end for
8:      else
9:          for $i \leftarrow s - 7$ to $s$ do
10:             Perform Step 6 of Algorithm 5;
11:         end for
12:     end if
13:     if ($s < 6$) then
14:         for $i \leftarrow 0$ to $s$ do
15:             Perform Step 6 of Algorithm 5;
16:         end for
17:     else
18:         for $i \leftarrow 0$ to 6 do
19:             Randomly extract the frame $i$ in the final keyframe sequence D and perform Step 6 of Algorithm 5;
20:         end for
21:     end if
22:     $s \leftarrow s + 1$; // Add the frame $r_i$ to the final keyframe sequence D;
23: end for
24: Perform Step 19 of Algorithm 4;
25: return The optimized poses in the keyframe sequence D.
```

plug-and-play, runs independently, and feeds the results to the next module directly.

IV. EXPERIMENTS

In Section IV-A, two publicly available datasets are presented to train and test DEM. In Section IV-B, online data augmentation is shown, training DEM for further accurate depth prediction. In Section IV-C, we preprocess the datasets to obtain different modalities of inputs for algorithms. In Section IV-D, error metrics are discussed.

A. DATASETS

In this section, we describe the indoor NYU-Depth-v2 and outdoor KITTI datasets for experiments.

1) NYU-Depth-v2 DATASET

The NYU-Depth-v2 dataset is used to train and assess models. The dataset includes 120,000 pairs of RGB and depth images captured by Microsoft Kinect v1 sensors. The largest ranging distance of Microsoft Kinect v1 is 10 meters. The dataset contains 464 scenes. In general, 249 scenes are adopted for model training and 215 scenes are employed for model testing. Following the classic work [8], 47,584 pairs of RGB-depth maps on the training dataset are leveraged for model training, and the testing dataset containing 654 pairs of RGB-depth images is used to test models. For a fair comparison, the same testing dataset is adopted to evaluate all models in this paper.

2) KITTI ODOMETRY DATASET

The KITTI dataset is adopted to further evaluate DEM. The KITTI contains depth and RGB images of urban, rural, and highway scenes. The depth images are captured by LiDARs. The maximum ranging distance of depth images is 100 meters. The odometry benchmark on KITTI includes 22 stereo sequences: 11 sequences are used for model training, and the others are for model testing. The resolution of training and testing images is 376×1241 and 370×1226, respectively. Following the research [8], we train DEM with 46k RGB-depth pairs from the training sequences and assess DEM with random 3200 RGB maps in the testing sequence. The testing dataset on KITTI is also leveraged by other models for evaluation in this paper.

For a fair comparison, we process the training and testing sequences on KITTI following the method [8]. First, LiDAR measurements are projected on corresponding RGB images from training and testing datasets, respectively. Second, we remove sky parts in raw images on training and testing datasets because measurements of the sky parts are meaningless. Then, the training and testing RGB-depth images of size 240×1200 are obtained. Third, online data augmentation is performed for the training dataset. The online data augmentation is described as follows.

B. DATA AUGMENTATION

To further achieve accurate depth estimation, we adopt online data augmentation for model training. The online data augmentation does not increase the number of images on training datasets. Following the work [8], the online data augmentation consists of five steps.

- Flip: we flip depth and RGB images horizontally with a 50% probability.
- Scale: we scale up and down depth and RGB images through a random scaling factor $r \in [1.0, 1.5]$.
- Translation: we translate depth and RGB images in a vertical direction.
- Rotation: we rotate depth and RGB images at the random angle $a \in [-4.5, 4.5]$.
- Color Normalization: we normalize RGB images by mean subtraction and division.

After color normalization, RGB and depth images are cropped from the center to obtain training images of the same size as that of the method [8]. Specifically, on the NYU-Depth-v2 dataset, training images of size 228×304 are obtained through data augmentation. To ensure the input size of model inference is consistent with that of model training, we crop from the center of RGB maps on the NYU-Depth-v2 dataset to acquire testing images of size 228×304. On the KITTI dataset, training images of size 228×912 are obtained by data augmentation. To ensure the input size of DEM is consistent for training and inference, we crop from the center of RGB images on the KITTI dataset to obtain testing images of size 228×912 for inference.

C. THREE MODALITIES OF INPUTS

After cropping images in Section IV-B, the obtained training and testing images on the same dataset are processed to generate three modalities of model inputs, i.e., RGBd data containing sparse depth and RGB images, sparse depth, and RGB images. The modalities of inputs are the same in model training and inference processes.

- Inputs of RGBd data: RGBd inputs are processed for model training and inference in two steps. 1) A few valid depth points are uniformly and randomly sampled from depth images on training and testing datasets, respectively. For a fair comparison, the number of valid depth samples is set to the same as that of research [8], [37]. 2) RGB images and corresponding depth samples on training and testing datasets are fused as RGBd data to train and test models respectively. The RGBd data on the same testing datasets are employed to evaluate models.
- Inputs of sparse depth: the same steps as those of RGBd data are performed to generate sparse depth inputs on training and testing datasets. The probability of depth samples at each position is about identical in a depth map because we sample depth points uniformly and randomly. The number of valid depth samples is set to the same as that of work [8], [37], [38] for a fair comparison. In each depth image, the largest number of valid depth samples is 200, which accounts for 0.04% and 0.06% of the total on the KITTI and NYU-Depth-v2 images, respectively. The above sampling proportion suggests that our depth sample-based inputs are sparse.
- Inputs of RGB images: RGB images captured by cameras or given by public datasets are used as inputs of model training and testing. For a fair comparison, the RGB images are the same to test DEM and other methods. Compared with depth estimation using RGBd data or sparse depth, predicting depth from a single RGB image is challenging because scale ambiguity exists in monocular depth estimation. Our DEM alleviates the ambiguity and estimates pixel-level depth maps reliably in outdoor or indoor scenarios.

D. ERROR METRICS

To evaluate DEM quantitatively, we use the error metrics which are also adopted by studies [1], [4]–[6], [7], [8], [15], [16], [24]–[26], [27], [28], [35], [37]–[39], [43]. The error metrics consider global statistics between a ground-truth depth image Y containing N depth pixels and a corresponding predicted depth map P consisting of N depth pixels. These error metrics are absolute relative difference (REL), root mean square error (RMSE), and δ_m, which are

- $REL = \frac{1}{|N|}\sum_{i,j}\frac{|y_{i,j}-p_{i,j}|}{p_{i,j}}$,
- $RMSE = \sqrt{\frac{1}{|N|}\sum_{i,j}\left|y_{i,j}-p_{i,j}\right|^2}$,
  where j and i are the ordinate and abscissa of a pixel in a depth map; $p_{i,j}$ and $y_{i,j}$ denote a predicted and ground-truth depth value of a pixel respectively, and
- $\delta_m = \frac{\mathbf{card}\left(\left\{p_{i,j}:\max\left\{\frac{p_{i,j}}{y_{i,j}},\frac{y_{i,j}}{p_{i,j}}\right\}<1.25^m\right\}\right)}{\mathbf{card}(\{y_{i,j}\})}$,
  where δ_m represents a percentage of estimated pixels in which relative errors are in a threshold [8], and $m = 1$, 2, or 3. Here, i and j are the abscissa and ordinate of

a pixel in a depth map. The depth value of a pixel in ground-truth and predicted depth maps is denoted as $y_{i,j}$ and $p_{i,j}$, respectively. The **card** is the cardinality of a set.

V. EVALUATING DEM

The performance of DEM is compared with state-of-the-art studies. The encoder, decoder, and loss function of DEM are evaluated on the NYU-Depth-v2 dataset in Section V-A. In Section V-B, to illustrate the effectiveness and certain generalization of DEM, we evaluate it on the NYU-Depth-v2 and KITTI datasets. In Section V-C, the inference latency of DEM is evaluated on different platforms. The computation resources of DEM are assessed on embedded devices in Section V-D.

Development Environment: In this section, models are trained in the cloud platform equipped with NVIDIA Tesla P100 graphics card and Intel i9 CPU. Following the work [8], the batch size and epochs are set to 16 and 20 when DEM is trained. The model inference is performed in the cloud platform and embedded platform (NVIDIA Jetson TX2). The type of inputs is the same for model training and inference.

A. *EVALUATING ENCODERS, DECODERS, AND LOSS FUNCTIONS*

In this section, we demonstrate that our encoder, decoder, and loss function are effective.

1) EVALUATING ENCODERS

As presented in Table 1, encoders are assessed. The same RGB input images, decoders, and loss functions are adopted in the same group. The loss function $\mathcal{L}_1$ is chosen for model training. The decoder in different groups is fixed as Deconv3 for a fair comparison. Deconv3 denotes transposed convolution layers with a 3×3 kernel. Here, ResNet-18 represents the encoder containing a convolution (Conv) layer, batch normalization (BN) layer, and ResNet-18. Similarly, the following CNN networks are expressed in the same way. The REL and RMSE are the-lower-the-better, and δ_m ($m = 1$, 2, or 3) is the-higher-the-better. Then, RMSE is analyzed as a representative.

TABLE 1. Evaluating encoders on the NYU-Depth-v2 dataset.

Input	Encoder	Decoder	Loss	RMSE	REL	δ_1	δ_2	δ_3
RGB	DPN-68	Deconv3	$\mathcal{L}_1$	0.528	0.143	80.9	95.2	98.7
	DPN-131	Deconv3	$\mathcal{L}_1$	0.519	0.145	81.1	95.8	99.0
	Resnet-18	Deconv3	$\mathcal{L}_1$	0.561	0.162	77.9	94.6	98.4
	Encoder [1]	Deconv3	$\mathcal{L}_1$	0.558	0.161	78.1	93.6	98.0
	Encoder [8]	Deconv3	$\mathcal{L}_1$	0.533	0.151	79.0	95.4	98.8
	Our Encoder	Deconv3	$\mathcal{L}_1$	**0.497**	**0.136**	**82.4**	**96.1**	**99.0**
	Encoder [16]	Deconv3	$\mathcal{L}_1$	0.599	0.178	74.2	92.1	96.9
	DenseNet-121	Deconv3	$\mathcal{L}_1$	0.519	0.143	81.5	95.6	98.8

As listed in Table 1, our encoder is more accurate than state-of-the-art encoders on the NYU-Depth-v2 dataset. We highlight better results in bold. The results [1], [8], [16] are obtained by implementing authors' respective codes. In the work [16], the encoder-decoder model "MobileNet-NNConv5dw" is employed in this paper for comparison. We perform models including DPN-68, DPN-131, ResNet-18, or DenseNet-121 and acquire their results. Compared with models including DPN-68, DPN-131, ResNet-18, or DenseNet-121, the RMSE of our encoder obtains at least 4.2% improvement. Compared with models [1], [8], [16], our RMSE is 6.8% lower. These results quantitatively justify that our encoder is more suitable than others when predicting depth.

2) EVALUATING DECODERS

As shown in Table 2, our decoder provides more reliable depth than state-of-the-art decoders. All models use RGB inputs on the NYU-Depth-v2 dataset for model training and inference. The results of models [5], [24] are taken from their respective papers. The results of studies [8], [16] are obtained by running their corresponding models. We implement the models including ResNet-18 and acquire their results. In Row 2-4, our decoder is 0.5% more accurate than Deconv3 and UpProj in RMSE. In Row 5-10, UpProj is slightly better than UpConv. The results are consistent with those in the work [8]. In the table, our RMSE is always lower than that of research [5], [8], [16], [24] when the proposed decoder is employed. As expected, our decoder outperforms other decoders in accuracy.

TABLE 2. Evaluating decoders on the NYU-Depth-v2 dataset.

Input	Encoder	Decoder	Loss	RMSE	REL	δ_1	δ_2	δ_3
RGB	ResNet-18	Deconv3	$\mathcal{L}_1$	0.561	0.162	77.9	94.6	98.4
	ResNet-18	UpProj	$\mathcal{L}_1$	0.547	0.156	79.1	94.9	98.6
	ResNet-18	Our Decoder	$\mathcal{L}_1$	**0.544**	**0.154**	**79.4**	**95.0**	**98.6**
RGB	Model [24]	Model [24]	Loss [24]	0.641	0.158	76.9	95.0	98.8
	Encoder [5]	UpConv	$\mathcal{L}_2$	0.606	**0.139**	77.8	94.4	98.5
	Encoder [16]	Decoder [16]	$\mathcal{L}_1$	0.582	0.165	76.8	93.6	98.0
	Encoder [8]	UpConv	$\mathcal{L}_1$	0.529	0.149	79.4	95.5	98.9
	Encoder [8]	UpProj	$\mathcal{L}_1$	0.528	0.144	**80.3**	95.2	98.7
	Encoder [8]	Our Decoder	$\mathcal{L}_1$	**0.525**	0.143	79.4	**95.5**	**98.9**

In summary, our decoder has three advantages. First, the decoder outputs high-resolution depth maps. Second, the decoder recovers local features, extracts features, and learns features through learnable transposed convolution and convolution networks. In contrast, the local features are easily lost by linear interpolation in decoders such as [5], [8]. Third, properly matched different encoders, the decoder is convenient to improve depth prediction accuracy. The proposed encoder and decoder compose the depth estimation model (DEM). Without additional pre-processing or post-processing steps, we train the end-to-end DEM with the proposed loss function.

3) EVALUATING LOSS FUNCTIONS

As shown in Table 3, our loss function provides comparable accuracy. The results of studies [5], [8], [16], [37] are obtained by implementing their corresponding models. We perform models including ResNet-18 and ResNet-34, and acquire their results. The NYU-Depth-v2 dataset is leveraged for model training and inference. The modalities of inputs are the same for model training and inference.

In all groups of Table 3, our loss function outperforms state-of-the-art loss functions. For example, the RMSE of our loss function is at least 0.8% lower than that of $\mathcal{L}_1$, $\mathcal{L}_2$, and berHu, in the RGB group where inputs are RGB images for model training and testing. The results in the sd group also confirm that our loss function L is effective in depth estimation. Here, in the sd group, inputs of model training and testing are sd-200 (200 valid depth points generated by random sampling in each image) on the NYU-Depth-v2 dataset.

TABLE 3. Evaluating loss functions on the NYU-Depth-v2 Dataset.

Input	Encoder	Decoder	Loss	RMSE	REL	δ_1	δ_2	δ_3
RGB	ResNet-18	Deconv$_3$	$\mathcal{L}_2$	0.575	0.170	76.1	94.0	98.4
	ResNet-18	Deconv$_3$	$\mathcal{L}_1$	0.561	0.162	77.9	94.6	98.4
	ResNet-18	Deconv$_3$	Our Loss	0.559	0.159	78.2	94.6	98.6
	Our Encoder	Our Decoder	$\mathcal{L}_2$	0.514	0.145	80.7	96.0	99.0
	Our Encoder	Our Decoder	$\mathcal{L}_1$	0.494	0.134	82.9	96.2	99.0
	Our Encoder	Our Decoder	Our Loss	**0.490**	0.135	**82.9**	**96.3**	**99.1**
	Encoder [8]	Deconv$_2$	$\mathcal{L}_2$	0.610	0.185	71.8	93.4	98.3
	Encoder [16]	Deconv$_2$	$\mathcal{L}_1$	0.594	0.176	75.6	93.7	97.8
	Encoder [8]	Deconv$_2$	$\mathcal{L}_1$	0.552	0.159	77.5	95.0	98.7
	Encoder [8]	Deconv$_2$	Our Loss	0.550	0.158	77.6	95.1	98.7
	Encoder [16]	UpProj [8]	$\mathcal{L}_1$	0.582	0.163	76.8	93.7	98.1
	Encoder [5]	UpProj [8]	berHu	0.573	**0.127**	81.1	95.3	98.8
	Encoder [5]	UpProj [8]	Our Loss	0.523	0.142	82.0	95.7	98.8
sd	ResNet-34	UpConv	$\mathcal{L}_2$	0.238	0.051	96.6	99.3	99.8
	ResNet-34	UpConv	$\mathcal{L}_1$	0.237	0.046	96.9	99.3	99.8
	ResNet-34	UpConv	Our Loss	0.235	0.047	96.8	99.3	99.8
	Our Encoder	Our Decoder	$\mathcal{L}_2$	0.239	0.053	97.0	**99.4**	**99.9**
	Our Encoder	Our Decoder	$\mathcal{L}_1$	0.231	0.043	97.0	99.3	99.8
	Our Encoder	Our Decoder	Our Loss	**0.223**	**0.041**	**97.1**	99.3	99.8
	Encoder [37]	Decoder [37]	$\mathcal{L}_1$	0.390	0.100	92.1	98.1	-
	Encoder [37]	Decoder [37]	Our Loss	0.380	0.061	93.1	98.5	-
	Encoder [16]	UpProj [8]	$\mathcal{L}_1$	0.286	0.061	96.3	99.1	99.8
	Encoder [8]	UpProj [8]	$\mathcal{L}_1$	0.259	0.054	96.3	99.2	99.8
	Encoder [8]	UpProj [8]	Our Loss	0.236	0.047	97.0	99.3	99.8

Consequently, when the proposed loss function L is adopted, the accuracy of depth estimation is higher than that of the state of the arts. Our accurate results in Table 3 benefit from the use of relative relationships by the proposed loss function L. The relationships among sparse depth samples encourage prediction to agree with ground truth. The ablation experiments of our encoder, decoder, and loss functions verify that they all enable finer depth estimation. We will compare the encoder-decoder model DEM with other studies in the next section.

B. COMPARISON WITH STATE OF THE ARTS

In this section, we verify the performance of DEM and discuss prediction results on two benchmark datasets.

1) COMPARISON ON THE NYU-Depth-v2 DATASET

Different models are evaluated on the NYU-Depth-v2 testing dataset. In Table 4 and Table 5, the results of models [1], [5]–[7], [8], [15], [24], [25], [28], [32], [36], [39], [43] are taken from their respective papers. The results of models [16], [33], [37] are acquired by implementing their respective methods. In the work [32], we select the first training setup because it consumes less calculation than other training setups. In the research [33], we adopt the original parameter setup to obtain better results than other parameter setups. The "w/o pre-trained weights" in Table 4 and Table 5 represents that our encoder is initialized without pre-trained weights in model training. The "semi" denotes that the study [39] takes semi-dense maps as inputs.

TABLE 4. Comparison using RGB/RGBd inputs on the NYU-Depth-v2 dataset.

Input	#Samples	Method	RMSE	REL	δ_1	δ_2	δ_3
RGB	0	Gur and Wolf [32]	0.766	0.254	69.1	88.0	94.4
	0	Roy and Todorovic [25]	0.744	0.187	-	-	-
	0	Eigen and Fergus [24]	0.641	0.158	76.9	95.0	98.8
	0	Xu *et al.* [15]	0.593	0.125	80.6	95.2	98.6
	0	Xu *et al.* [6]	0.586	0.121	81.1	95.4	98.7
	0	Wofk *et al.* [16]	0.583	0.164	76.7	93.7	98.0
	0	Xu *et al.* [43]	0.582	0.120	81.7	95.4	98.7
	0	Xu *et al.* [28]	0.579	**0.108**	82.3	95.7	98.7
	0	Laina *et al.* [5]	0.573	0.127	81.1	95.3	98.8
	0	Tu *et al.* [1]	0.547	0.152	79.6	94.8	98.5
	0	Ma and Karaman [8]	0.514	0.143	81.0	95.9	98.9
	0	DEM	**0.490**	0.135	**82.9**	**96.3**	**99.1**
	0	w/o pre-trained weights	0.637	0.187	72.1	91.5	97.5
	100	Wang + Eigen [36]	0.502	-	80.5	94.8	98.8
	225	Liao *et al.* [7]	0.442	0.104	87.8	96.4	98.9
	20	Hu *et al.* [33]	0.526	-	-	-	-
	20	Wofk *et al.* [16]	0.385	0.086	92.0	97.9	99.2
	20	Ma and Karaman [8]	0.351	0.078	92.8	98.4	99.6
	20	DEM	**0.314**	**0.069**	**94.3**	**98.8**	**99.7**
	50	Hu *et al.* [33]	0.495	-	-	-	-
	50	Wofk *et al.* [16]	0.314	0.065	94.8	98.6	99.5
	50	Ma and Karaman [8]	0.281	0.059	95.5	99.0	99.7
	50	DEM	**0.248**	**0.050**	**96.6**	**99.3**	**99.8**
	200	Hu *et al.* [33]	0.492	-	-	-	-
	200	Wofk *et al.* [16]	0.292	0.068	95.8	99.3	99.8
	200	Li *et al.* ($\mathcal{L}_2$ loss) [37]	0.943	0.572	**99.5**	99.6	-
	200	Li *et al.* ($\mathcal{L}_1$ loss) [37]	0.256	0.046	98.3	**99.7**	-
	200	Ma and Karaman [8]	0.230	0.044	97.1	99.4	99.8
	200	DEM	**0.194**	**0.036**	98.0	99.6	**99.9**
	200	w/o pre-trained weights	0.226	0.042	97.3	99.4	99.8
	semi	Yang *et al.* [39]	0.243	0.064	96.6	99.6	99.9

In Table 4, the results denote that DEM achieves considerable improvement as compared with other approaches. Specifically, our RMSE is respectively 10.4%, 14.5%, 16.4%, 4.6%, 17.4%, 23.6%, 34.1%, 16.0%, 15.4%, 36.0%, and 15.8% lower than that of research [1], [5], [6], [8], [15], [16], [24], [25], [28], [32], [43] when inputs are RGB images for model training and evaluation. When model inputs are RGB images and (20, 50, or 200) sparse samples, DEM is also more accurate than others. Additionally, the encoder initialized with pre-trained weights outperforms that without pre-trained weights in DEM. The above contrast confirms that our encoder, decoder, and loss function are effective in depth estimation when inputs are RGB images or RGBd data (RGB images and sparse samples).

In Table 5, we use sd-20, sd-50, and sd-200 as model inputs. The sd-20 denotes inputs contain only 20 valid depth samples for model training and inference. The sd-50 and sd-200 are expressed in the same way. The depth estimation quality improves as the proportion of depth samples increases. All accuracy of DEM is comparable as compared with the state of the arts. For example, our RMSE is respectively 3.1%, 3.9%, 4.1%, 67.6%, 34.9%, and 1.3% lower than that of methods [1], [8], [16], [33], [37], [38] when model inputs are sd-20. The accuracy of DEM is also higher than that of other studies when model inputs are sd-50. Our RMSE is respectively 13.9%, 22.8%, 82.4%, and 42.8% better than that of work [8], [16], [33], [37] when model inputs are sd-200.

TABLE 5. Comparison using sparse depth inputs on the NYU-Depth-v2 dataset.

Input	#Samples	Method	RMSE	REL	δ_1	δ_2	δ_3
sd	20	Hu *et al.* [33]	1.369	-	-	-	-
	20	Li *et al.* ($\mathcal{L}_1$ loss) [37]	0.680	0.240	70.0	86.5	-
	20	Wofk *et al.* [16]	0.462	0.106	88.0	96.3	98.8
	20	Ma and Karaman [8]	0.461	0.110	87.2	96.1	98.8
	20	Tu *et al.* [1]	0.457	0.107	87.5	96.3	98.8
	20	Huang *et al.* [38]	0.449	0.110	-	-	-
	20	DEM	**0.443**	**0.100**	**88.0**	**96.4**	**98.8**
	50	Hu *et al.* [33]	1.310	-	-	-	-
	50	Wofk *et al.* [16]	0.350	0.075	93.1	98.2	99.4
	50	Li *et al.* ($\mathcal{L}_1$ loss) [37]	0.440	0.130	83.0	95.1	-
	50	Ma and Karaman [8]	0.347	0.076	92.8	98.2	99.5
	50	DEM	**0.342**	**0.070**	**93.2**	**98.2**	**99.5**
	200	Hu *et al.* [33]	1.265	-	-	-	-
	200	Li *et al.* ($\mathcal{L}_1$ loss) [37]	0.390	0.100	92.1	98.1	-
	200	Wofk *et al.* [16]	0.289	0.062	96.5	99.3	99.8
	200	Ma and Karaman [8]	0.259	0.054	96.3	99.2	99.8
	200	DEM	**0.223**	**0.041**	**97.1**	**99.3**	**99.8**
	200	w/o pre-trained weights	0.230	0.043	96.0	99.0	99.7

In Fig. 4, the visual results also show that DEM boosts the accuracy of depth prediction. Here, depth maps are predicted with RGB images or sd-200 inputs on the NYU-Depth-v2 dataset. The modalities of inputs are the same in training and inference processes. The results of Ma and Karaman [8] are taken from their paper. The difference between methods is marked by boxes or circles in Fig. 4. In each group, our depth maps are clearer than others. Specifically, the objects predicted by DEM have distinct edges and corners. The estimated details of a sofa, chair, and table are easy to distinguish in Fig. 4(b). The sofa, table, and mural in Fig. 4(d) also have complete contours. In contrast, the sofa and table predicted by the work [8] are unclear, as listed in the first and second columns of Fig. 4(c). The objects [8] in Fig. 4(e) are still

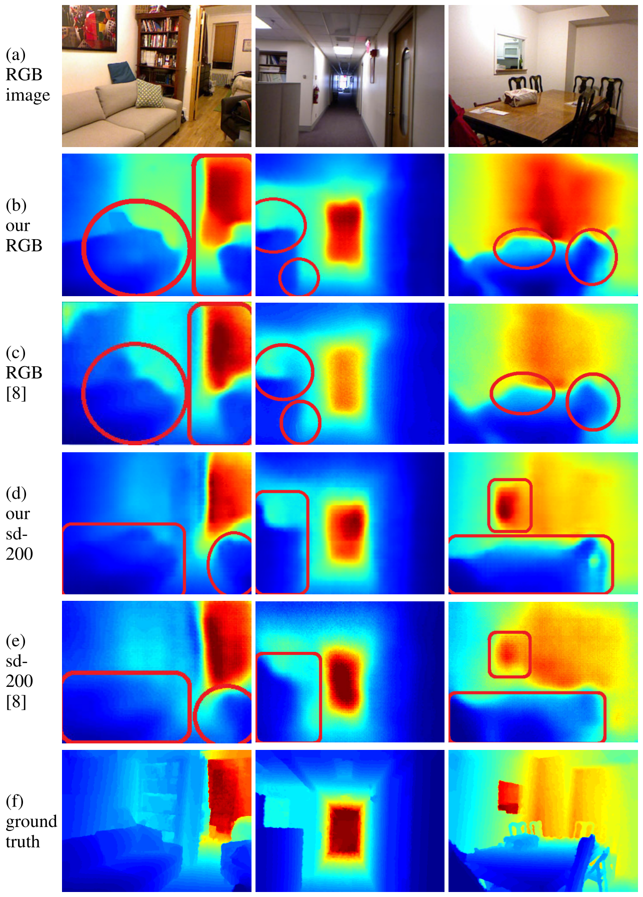


FIGURE 4. Predictions on the NYU-Depth-v2 dataset. From top to bottom: (a) monocular RGB images; (b) our estimation relying on RGB images; (c) RGB-based predictions [8]; (d) our estimation relying on sd-200 data; (e) the estimation in the work [8] based on sd-200 data; (f) ground truth.

ambiguous. These visual results further confirm that DEM considerably improves depth estimation.

2) COMPARISON ON THE KITTI DATASET

On the outdoor KITTI testing dataset, we evaluate DEM and compare it with the state of the arts, as shown in Table 6. The KITTI dataset is challenging because the largest ranging distance of the KITTI dataset is ten times that of NYU-Depth-v2. The input modalities in model training are the same as those in model inference. The results of models [4], [7], [8], [26], [27], [32], [35], [36] are taken from their respective papers. The result of the research [22] is taken from the paper [4]. The results of models [16], [33] are acquired by implementing their respective methods.

In Table 6, DEM performs better than other methods. For example, our RMSE is respectively 38.1%, 29.3%, 49.2%, 29.6%, 14.6%, 41.0%, 14.5%, and 18.5% lower than that of models [4], [7], [8], [16], [26], [32], [33], [35] when model inputs are RGB images. The RMSE of DEM is 65.2% better than that of Cadena *et al.* [35], and our depth samples are less than those of the method [35] when model inputs are RGB images and 500 depth samples. Additionally, our output

TABLE 6. Comparison results on the KITTI dataset.

Input	#Samples	Method	RMSE	REL	δ_1	δ_2	δ_3
RGB	0	Saxena *et al.* [22]	8.734	0.280	60.1	82.0	92.6
	0	Mancini *et al.* [27]	7.508	-	31.8	61.7	81.3
	0	Eigen *et al.* [4]	7.156	0.190	69.2	89.9	96.7
	0	Wang *et al.* [26]	6.298	0.185	75.2	91.3	96.2
	0	Ma and Karaman [8]	6.266	0.208	59.1	90.0	96.2
	0	Hu *et al.* [33]	5.437	-	-	-	-
	0	Wofk *et al.* [16]	5.191	0.145	84.3	95.2	98.0
	0	Gur and Wolf [32]	5.187	0.141	84.6	95.3	98.1
	0	DEM	**4.433**	**0.101**	**88.2**	**96.1**	**98.4**
	~650	Cadena *et al.* [35]	7.140	0.179	70.9	88.8	95.6
	500	Hu *et al.* [33]	5.389	-	-	-	-
	500	Wang + Eigen [36]	5.140	-	75.1	91.7	97.0
	225	Liao *et al.* [7]	4.500	0.113	87.4	96.0	98.4
	500	Ma and Karaman [8]	3.378	0.073	93.5	97.6	98.9
	500	Wofk *et al.* [16]	3.033	0.051	95.3	97.9	99.0
	500	DEM	**2.485**	**0.040**	**96.4**	**98.3**	**99.2**

TABLE 7. Comparison of generalization error on the Make3D dataset.

Input	#Samples	Method	RMSE	REL	δ_1	δ_2	δ_3
RGB	0	Baseline [22]	-	0.698	-	-	-
	0	Saxena *et al.* [21]	16.700	0.530	-	-	-
	0	Wofk *et al.* [16]	10.281	0.594	16.1	42.6	74.2
	0	DEM	**10.003**	**0.529**	**18.2**	**45.7**	**79.8**
	500	Wofk *et al.* [16]	5.658	0.135	87.8	94.2	96.7
	500	Ma and Karaman [8]	5.525	0.140	86.5	94.2	96.8
	500	DEM	**5.455**	**0.104**	**89.4**	**95.0**	**97.2**

TABLE 8. Runtime of DEM.

Dataset	Platform	RGBd-50 (ms)	sd-50 (ms)	RGB (ms)
NYUDepthv2 (480×640)	Tesla P100	17	15	16
	Jetson TX2	112	109	111
KITTI (370×1226)	Tesla P100	19	18	18
	Jetson TX2	120	118	119

size of DEM is 48.1 times that of the method [35]. Hence, more accurate and high-resolution depth maps are predicted with DEM than other models. The RMSE of DEM is reduced by 23.7% compared with the research [7]. In addition, DEM has 55.6% fewer depth samples than the work [7]. Relative to the state of the arts [8], [16], [33], [36], DEM also has higher accuracy. These results demonstrate the encoder, decoder, and loss function of DEM are more precise than others.

3) COMPARISON OF GENERALIZATION ERROR ON THE Make3D TESTING DATASET.

To verify the generalization of DEM, we train DEM on one dataset such as KITTI and test it on another dataset like Make3D. For comparison, the baseline is selected from the paper [22]. The results of Saxena *et al.* are taken from their paper [21]. Similar to DEM, the state of the arts [8], [16] are trained on KITTI and tested on Make3D. When model inputs are RGB images, the REL of DEM is 24.2% lower than that of the baseline [22]. The RMSE of DEM is respectively 40.1% and 2.7% better than that of methods [16], [21]. When model inputs are RGB images and 500 sparse samples, our RMSE is respectively 12.7% and 35.9% lower than that of the studies [8], [16]. Different from DEM, the classic work [21] and the baseline [22] use Make3D to train and test models, but DEM performs better than them. These results show that DEM has a certain generalization ability.

C. EVALUATING RUNTIME ON DIFFERENT PLATFORMS

The inference time of DEM is evaluated in the cloud platform and embedded platform. The inference time is measured three times, and then the average of inference time of each image is computed. The inputs on NVIDIA Jetson TX2 are the same as those in the cloud platform for training and inference. In the sd group of Table 8, sd-50 denotes model inputs containing only 50 valid depth samples in an image for training and inference. In the RGBd group, RGBd-50 means the inputs include RGB images and 50 valid depth samples for training and evaluation. All values are in milliseconds (ms) in Table 8. The accuracy metrics are not shown here because they have been listed in Table 4 and 5. The accuracy has no difference in the cloud platform and embedded platform when inputs and models are the same.

Table 8 shows that DEM satisfies real-time (9 frames per second) constraints when inputs are RGB images of size 480 × 640 on TX2. Specifically, in the RGBd group, the runtime of the RGBd group is the longest in all groups on TX2 or in the cloud platform because DEM predicts depth with sparse depth points and RGB images. In the sd group, the short runtime is spent whether in the cloud or on TX2, owing to only valid depth points used as inputs of DEM. In the RGB group, the runtime is better than that of the RGBd group. Here, the runtime of DEM on the KITTI dataset is a little longer than that on the NYU-Depth-v2 dataset because the size of inputs on the KITTI dataset is larger than that on the NYU-Depth-v2 dataset. The runtime of RGB input images of size 480 × 640 meets real-time requirements on TX2. This result is reasonable since the RGB group predicts depth maps with RGB inputs alone. Additionally, RGB images are easily obtained from low-cost monocular cameras. Therefore, RGB images are chosen as inputs of DEM to test computation resources on embedded devices.

D. EVALUATING COMPUTATION RESOURCES ON EMBEDDED DEVICES

To evaluate computation resources of DEM on the representative embedded device NVIDIA Jetson TX2, we measure metrics such as the number of parameters, memory utilization, CPU/GPU consumption with their frequency, power consumption, and energy consumption, as listed in Table 9. These metrics are widely used to evaluate computation resources of algorithms on embedded systems [44], [45]. The result of ResNet-FC-160 × 128 [5] is acquired by implementing the method ResNet-FC-160×128 on TX2. The

TABLE 9. Evaluating computation resources on embedded devices.

Inputs	Encoder	Decoder	Parameters	Memory Utilization	CPU Consumption Frequency	GPU Consumption Frequency	Power	Energy	Time (ms)
RGB	Ours	Ours	**4.5×10^7**	29%	21%/1205MHz	**76%/978MHz**	**6.9W**	**503.8J**	**72594.0**
	ResNet-50 [8]	UpProj [8]	6.3×10^7	**28%**	**20%/942MHz**	81%/981MHz	7.4W	1548.7J	208626.0
	ResNet-50 [5]	FC-160×128 [5]	3.6×10^8	50%	22%/1216MHz	82%/989MHz	7.9W	758.5J	95764.0
	ResNet-50 [34]	Feature Fusion [34]	11.1×10^7	65%	23%/1345MHz	86%/990MHz	9.8W	1653.7J	209012.0
RGBd-200	Ours	Ours	**4.5×10^7**	29%	21%/1206MHz	**76%/978MHz**	**7.2W**	**585.6J**	**81096.0**
	ResNet-50 [8]	UpProj [8]	6.3×10^7	**28%**	**20%/1060MHz**	85%/980MHz	7.6W	1595.5J	209280.0
	ResNet-50 [5]	FC-160×128 [5]	3.6×10^8	50%	22%/1223MHz	86%/990MHz	8.0W	808.1J	100764.0
	ResNet-50 [34]	Feature Fusion [34]	11.1×10^7	65%	23%/1359MHz	86%/993MHz	9.9W	1776.1J	210213.0

result of Ma and Karaman [8] is obtained by running their code on TX2.

For comparison, the computation resources of methods [5], [8] are evaluated under the same condition as that of DEM. When stably predicting depth maps of 654 RGB images (the NYU-Depth-v2 testing dataset) by using MAXP_CORE_ARM mode on TX2, we record the metrics of memory utilization, CPU/GPU consumption with their frequency, and power consumption, and we then compute average values of them. The power consumption is computed by multiplying voltage and current. The voltage and current are measured by a power supply. When completing depth estimation of 654 RGB images, we record the runtime and compute the energy consumption. The energy consumption is computed by multiplying the power consumption and runtime of 654 RGB images.

On all metrics in Table 9, parameters are the-less-the-better. The memory utilization, CPU/GPU consumption, CPU/GPU frequency, power consumption, and energy consumption are the-lower-the-better. The CPU/GPU frequency changes with CPU/GPU workloads dynamically. The percentage of CPU/GPU consumption is relative to the CPU/GPU frequency. Additionally, in the RGB group of Table 9, inputs of all methods are RGB images for model training and inference. In the RGBd group of Table 9, inputs of all methods are RGB images and sparse depth samples for model training and evaluation.

As presented in Table 9, DEM is suited to run on embedded devices. In contrast to the state of the art, DEM boosts the performance of depth prediction with low consumption of GPU, power, and energy. Specifically, in RGB and RGBd-200 groups, we perform better than those of the work [5], [8], [34] on metrics such as runtime and consumption of GPU, power, and energy. The memory utilization and CPU consumption in the research [8] slightly outperform ours. Also, DEM is less complex and has 28.6% fewer parameters than the algorithm [8]. These results demonstrate that DEM is good at reducing the runtime and consumption of GPU, power, and energy. Therefore, DEM is real-time with low consumption of GPU, power, and energy for fast depth estimation on embedded devices.

VI. APPLICATION OF DEM AND EVALUATION OF SLAM

In Section VI-A, we apply DEM to LiDAR super-resolution. The application result is qualitatively compared with the state of the art. In Section VI-B, we quantitatively assess the application of DEM in LiDAR super-resolution. In Section VI-C, we qualitatively evaluate the DEM-based SLAM. In Section VI-D, the DEM-based SLAM is quantitatively compared with well-known SLAM systems, and results are analysed in detail.

A. APPLYING DEM TO LiDAR SUPER-RESOLUTION

As shown in Fig. 5, we apply DEM to LiDAR super-resolution following the representative study [8]. On the KITTI dataset, RGBd-20000 data (20000 depth samples in each image) are used as inputs of models for training and inference. The results of Ma and Karaman [8] are acquired by running their code. The good application in LiDAR super-resolution represents easily recognizable objects in Fig. 5. In Fig. 5(c), faraway cars and trees are recognized by our estimation. In Fig. 5(b), the car and tree at a long distance cannot be identified. The object shape in our prediction is clear. For instance, nearby cars in Fig. 5(c) are distinguished easily. The window of the car is identified more clearly in our results than that in the research [8]. These significant differences between Fig. 5(b) and Fig. 5(c) imply that DEM is more reliable than the method [8]. Additionally, the estimation of DEM has fine contours of objects, compared with the ground-truth depth projected from LiDAR measurements in Fig. 5(d). Our experiments justify that DEM can be applied to LiDAR super-resolution well.

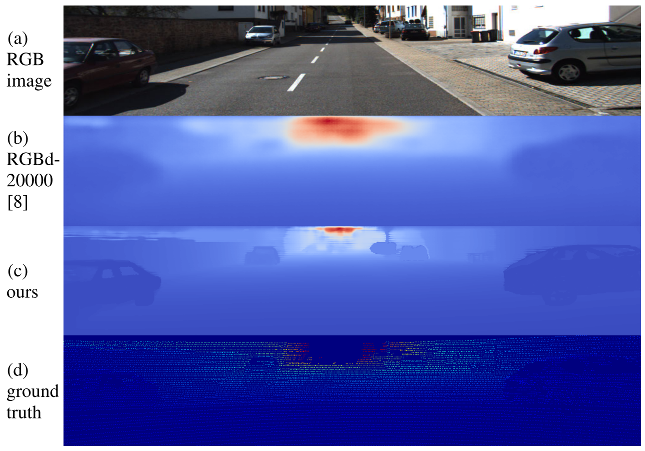


FIGURE 5. Applications in LiDAR super-resolution. Inputs of all algorithms are RGBd-20000 data for model training and inference. (a) RGB images; (b) dense depth prediction in the work [8] based on RGBd-20000 data; (c) our depth estimation relying on RGBd-20000 data; (d) ground-truth depth projected from LiDAR measurements.

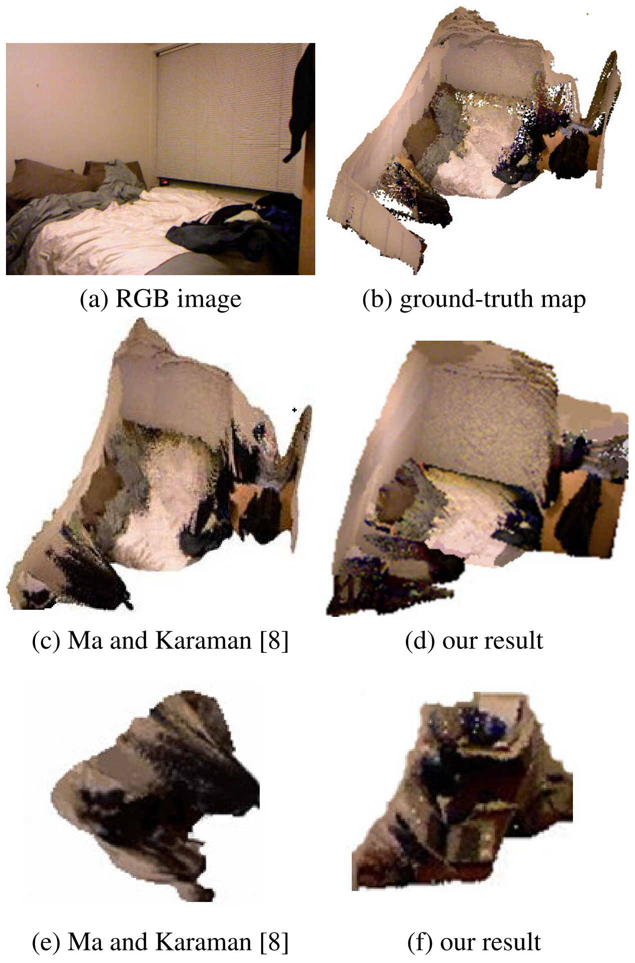


FIGURE 6. **Reconstruction results using the NYU-Depth-v2 testing scenes; (a) RGB images; (b) the ground-truth map; (c) the result of the research [8]; (d) the result of the DEM-based SLAM; (e) the result of the front-end of SLAM [8]; (f) the result of the front-end of DEM-based SLAM.**

B. EVALUATING APPLICATIONS QUANTITATIVELY

The mean absolute error (MAE) metric is computed to quantitatively evaluate applications of DEM in LiDAR super-resolution again. The MAE represents the average of absolute deviations between ground-truth and predicted depth maps. The MAE of DEM is 0.745 when the inputs for model training and testing are RGBd-500 data (500 depth samples in each image) on the KITTI dataset. With RGBd-500 inputs on KITTI for model training and testing, we run the source code [8]. Then, the MAE [8] is obtained (1.209). In contrast, our MAE is 38.4% lower than that of the study [8]. In Table 6, we also quantitatively demonstrated the application of DEM in super-resolution performs better than that of the work [8]. Specifically, DEM outperforms the state-of-the-arts on metrics of RMSE, REL, δ_1, δ_2, and δ_3 in Table 6. These assessments quantitatively show that applying DEM to LiDAR super-resolution is effective.

C. EVALUATING DEM-BASED SLAM QUALITATIVELY

As shown in Fig. 6, the DEM-based SLAM is evaluated qualitatively. The reconstruction results of SLAM are presented in Fig. 6(c) and Fig. 6(d). To demonstrate the local consistency in SLAM systems, the front-end of SLAM is shown in Fig. 6(e) and Fig. 6(f). The results in Fig. 6(c) and Fig. 6(e) are obtained by implementing the representative method [8]. Here, we use inputs of monocular RGB images on the NYU-Depth-v2 dataset for model training and testing. The outputs of DEM from RGB images are employed by our SLAM. In contrast, the work [8] employs inputs of RGB and sparse depth on the NYU-Depth-v2 dataset for model training and testing. The outputs of model testing relying on RGB and sparse depth are leveraged as inputs of the SLAM system [8].

In Fig. 6, the reconstruction results of the DEM-based SLAM are close to the ground truth, although using monocular RGB images as inputs. Compared with the state of the art [8] from RGB and sparse depth inputs, the DEM-based SLAM reconstructs the detail of walls from RGB images, as displayed in Fig. 6(d). By contrast, the work [8] has little reconstruction of walls in Fig. 6(c). The reconstruction of textureless regions like walls shows that the DEM-based SLAM is dependable. Additionally, more complete objects are shown in the front-end of our SLAM in Fig. 6(f) than those in Fig. 6(e). The favorable results in Fig. 6(d) and Fig. 6(f) benefit from our reliable DEM and SLAM.

To further justify the effectiveness of the DEM-based SLAM, we provide visual results of optimized poses obtained in the pose prediction module of our SLAM. The monocular camera is moved a distance and then returned to the vicinity of an initial position. The poses of the end and starting point are presented in Fig. 7(a) and Fig. 7(b). The pose deviation is large between the origin and end point without pose optimization in Fig. 7(b). This deviation is reduced by optimizing poses with RANSAC PNP initial values, as shown in Fig. 7(a). These results confirm our poses are accurate by using G2o optimization with RANSAC PNP initial values in the pose prediction module. The optimized poses boost the accuracy of the DEM-based SLAM.

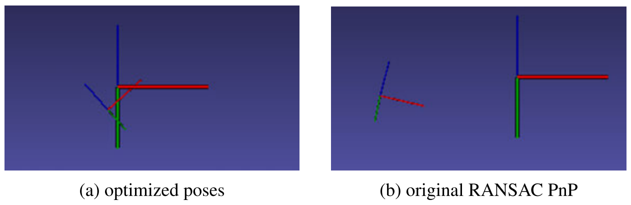


FIGURE 7. **Poses from SLAM. (a) The starting and ending pose predicted by G2o optimization with RANSAC PNP initial values; (b) The starting and ending poses estimated by the original RANSAC PnP.**

D. EVALUATING DEM-BASED SLAM QUANTITATIVELY

The TUM dataset[4] and Absolute Trajectory Error (ATE) RMSE (m) are adopted to test SLAM quantitatively. The ATE RMSE metric is computed as the RMSE between the ground truth and the estimated translation for each evaluated sequence. The ATE RMSE is low when SLAM performs well. As common evaluation metrics of SLAM, ATE RMSE is

[4]https://vision.in.tum.de/data/datasets/rgbd-dataset

widely leveraged in previous research [10]–[12], [17], [18], [19], [20].

On the public TUM dataset, the sequences of fr1_desk, fr3_long_office, and fr3_str_tex_far are chosen for SLAM evaluation. On these sequences, monocular RGB images are employed as inputs of the DEM-based SLAM. Here, DEM is trained on the NYU-Depth-v2 dataset and directly for inference on the TUM dataset. The outputs of DEM are used by the DEM-based SLAM for reconstruction.

The ATE RMSE of the DEM-based SLAM is compared with that of well-known SLAM [10]–[12], [17], [18], [19], [20] on the TUM sequences. The results are shown in Table 10. For comparison, the ATE RMSE of monocular ORB-SLAM [17] and RGBD ORB-SLAM2 [19] is obtained by running their code on the TUM sequences. The ATE RMSE of work [10]–[12], [18], [20] is taken from their respective papers.

In Table 10, the DEM-based SLAM performs well with near state-of-the-art accuracy. The results of RGBD ORB-SLAM2 are used as an upper bound of accuracy because RGBD ORB-SLAM2 yields small error relying on inputs of RGB images and ground-truth depth maps. In contrast to RGBD ORB-SLAM2, the DEM-based SLAM reconstructs scenes through only monocular RGB images, but the DEM-based SLAM outperforms deep learning-based SLAM methods [10]–[12], [20]. The extensive experiments demonstrate that the DEM-based SLAM is reliable.

TABLE 10. Absolute trajectory error RMSE (m) on TUM sequences.

Methods	fr3_str_tex_far	fr1_desk	fr3_long_office
Ours	0.018	0.020	0.021
Monocular ORB-SLAM [17]	0.843	0.080	1.206
CNN-SLAM [10]	0.214	-	0.542
Monocular SLAM [11]	0.105	0.192	0.635
LSD-SLAM [18]	0.870	0.062	1.207
GCN [12]	-	0.029	0.040
GCN-SLAM [20]	-	0.031	0.021
RGBD ORB-SLAM2 [19]	**0.010**	**0.016**	**0.010**

VII. CONCLUSION

In this paper, we propose the encoder-decoder model (DEM) to estimate depth, the loss function to guide the training of DEM, and the DEM-based SLAM system to reconstruct scenes with monocular cameras. The encoder reuses and re-exploits features effectively; the decoder recovers details that are generally lost by linear interpolation in other decoders; the loss function encourages estimated depth to agree with the ground truth. Our methods achieve comparable results on public datasets. Particularly, DEM outperforms state-of-the-art algorithms in accuracy, whether inputs are monocular images, sparse depth, or the fusion of both. Our DEM runs in nearly real-time (9 frames per second) and is efficient in the consumption of GPU, power, and energy on embedded devices. Additionally, the application of DEM in LiDAR super-resolution performs better than previous algorithms. Based on DEM, the proposed SLAM system reconstructs more accurate scenes than existing depth estimation-based SLAM. Future research will use lightweight depth estimation models to decrease the consumption of memory and CPU.

REFERENCES

[1] X. Tu, C. Xu, S. Liu, G. Xie, and R. Li, "Real-time depth estimation with an optimized encoder-decoder architecture on embedded devices," in *Proc. IEEE 21st Int. Conf. High Perform. Comput. Commun., IEEE 17th Int. Conf. Smart City, IEEE 5th Int. Conf. Data Sci. Syst. (HPCC/SmartCity/DSS)*, Aug. 2019, pp. 2141–2149.

[2] W. Hong, Z. Wang, M. Yang, and J. Yuan, "Conditional generative adversarial network for structured domain adaptation," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit.*, Jun. 2018, pp. 1335–1344.

[3] N. Chodosh, C. Wang, and S. Lucey, "Deep convolutional compressed sensing for LiDAR depth completion," in *Proc. Asian Conf. Comput. Vis. (ACCV)*, 2018, pp. 499–513.

[4] D. Eigen, C. Puhrsch, and R. Fergus, "Depth map prediction from a single image using a multi-scale deep network," in *Proc. Adv. Neural Inf. Process. Syst.*, 2014, pp. 2366–2374.

[5] I. Laina, C. Rupprecht, V. Belagiannis, F. Tombari, and N. Navab, "Deeper depth prediction with fully convolutional residual networks," in *Proc. 4th Int. Conf. 3D Vis. (3DV)*, Oct. 2016, pp. 239–248.

[6] D. Xu, E. Ricci, W. Ouyang, X. Wang, and N. Sebe, "Multi-scale continuous CRFs as sequential deep networks for monocular depth estimation," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jul. 2017, pp. 5354–5362.

[7] Y. Liao, L. Huang, Y. Wang, S. Kodagoda, Y. Yu, and Y. Liu, "Parse geometry from a line: Monocular depth estimation with partial laser observation," in *Proc. IEEE Int. Conf. Robot. Automat. (ICRA)*, May 2017, pp. 5059–5066.

[8] F. Ma and S. Karaman, "Sparse-to-dense: Depth prediction from sparse depth samples and a single image," in *Proc. IEEE Int. Conf. Robot. Automat. (ICRA)*, May 2018, pp. 4796–4803.

[9] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jun. 2016, pp. 770–778.

[10] K. Tateno, F. Tombari, I. Laina, and N. Navab, "CNN-SLAM: Real-time dense monocular SLAM with learned depth prediction," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jul. 2017, pp. 6565–6574.

[11] H. Luo, Y. Gao, Y. Wu, C. Liao, X. Yang, and K.-T. Cheng, "Real-time dense monocular SLAM with online adapted depth prediction network," *IEEE Trans. Multimedia*, vol. 21, no. 2, pp. 470–483, Feb. 2019.

[12] J. Tang, J. Folkesson, and P. Jensfelt, "Geometric correspondence network for camera motion estimation," *IEEE Robot. Autom. Lett.*, vol. 3, no. 2, pp. 1010–1017, Apr. 2018.

[13] Y. Chen, J. Li, H. Xiao, X. Jin, S. Yan, and J. Feng, "Dual path networks," in *Proc. Adv. Neural Inf. Process. Syst.*, 2017, pp. 4467–4475.

[14] G. Huang, Z. Liu, L. Van Der Maaten, and K. Q. Weinberger, "Densely connected convolutional networks," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jul. 2017, pp. 4700–4708.

[15] D. Xu, W. Wang, H. Tang, H. Liu, N. Sebe, and E. Ricci, "Structured attention guided convolutional neural fields for monocular depth estimation," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit.*, Jun. 2018, pp. 3917–3925.

[16] D. Wofk, F. Ma, T.-J. Yang, S. Karaman, and V. Sze, "FastDepth: Fast monocular depth estimation on embedded systems," in *Proc. IEEE Int. Conf. Robot. Automat. (ICRA)*, May 2019, pp. 6101–6108.

[17] R. Mur-Artal, J. M. M. Montiel, and J. D. Tardós, "ORB-SLAM: A versatile and accurate monocular SLAM system," *IEEE Trans. Robot.*, vol. 31, no. 5, pp. 1147–1163, Oct. 2015.

[18] J. Engel and T. Schöps, and D. Cremers, "LSD-SLAM: Large-scale direct monocular SLAM," in *Proc. Eur. Conf. Comput. Vis.*, 2014, pp. 834–849.

[19] R. Mur-Artal and J. D. Tardós, "ORB-SLAM2: An open-source SLAM system for monocular, stereo, and RGB-D cameras," *IEEE Trans. Robot.*, vol. 33, no. 5, pp. 1255–1262, Oct. 2017.

[20] J. Tang, L. Ericson, J. Folkesson, and P. Jensfelt, "GCNv2: Efficient correspondence prediction for real-time SLAM," *IEEE Robot. Autom. Lett.*, vol. 4, no. 4, pp. 3505–3512, Oct. 2019.

[21] A. Saxena, S. H. Chung, and A. Y. Ng, "3-D depth reconstruction from a single still image," *Int. J. Comput. Vis.*, vol. 76, no. 1, pp. 53–69, 2008.

[22] A. Saxena, M. Sun, and A. Y. Ng, "Make3D: Learning 3D scene structure from a single still image," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 31, no. 5, pp. 824–840, May 2009.

[23] K. Karsch, C. Liu, and S. B. Kang, "Depth transfer: Depth extraction from video using non-parametric sampling," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 36, no. 11, pp. 2144–2158, Nov. 2014.

[24] D. Eigen and R. Fergus, "Predicting depth, surface normals and semantic labels with a common multi-scale convolutional architecture," in *Proc. IEEE Int. Conf. Comput. Vis. (ICCV)*, Dec. 2015, pp. 2650–2658.

[25] A. Roy and S. Todorovic, "Monocular depth estimation using neural regression forest," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jun. 2016, pp. 5506–5514.

[26] A. Wang, Z. Fang, Y. Gao, X. Jiang, and S. Ma, "Depth estimation of video sequences with perceptual losses," *IEEE Access*, vol. 6, pp. 30536–30546, 2018.

[27] M. Mancini, G. Costante, P. Valigi, and T. A. Ciarfuglia, "Fast robust monocular depth estimation for obstacle detection with fully convolutional networks," in *Proc. IEEE/RSJ Int. Conf. Intell. Robots Syst. (IROS)*, Oct. 2016, pp. 4296–4303.

[28] D. Xu, E. Ricci, W. Ouyang, X. Wang, and N. Sebe, "Monocular depth estimation using multi-scale continuous CRFs as sequential deep networks," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 41, no. 6, pp. 1426–1440, Jun. 2019.

[29] D. Xu, W. Ouyang, X. Alameda-Pineda, E. Ricci, X. Wang, and N. Sebe, "Learning deep structured multi-scale features using attention-gated CRFs for contour prediction," in *Proc. Adv. Neural Inf. Process. Syst.*, 2017, pp. 3961–3970.

[30] Y. Cao, Z. Wu, and C. Shen, "Estimating depth from monocular images as classification using deep fully convolutional residual networks," *IEEE Trans. Circuits Syst. Video Technol.*, vol. 28, no. 11, pp. 3174–3182, Nov. 2018.

[31] L. He, C. Chen, T. Zhang, H. Zhu, and S. Wan, "Wearable depth camera: Monocular depth estimation via sparse optimization under weak supervision," *IEEE Access*, vol. 6, pp. 41337–41345, 2018.

[32] S. Gur and L. Wolf, "Single image depth estimation trained via depth from defocus cues," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jun. 2019, pp. 7683–7692.

[33] J. Hu, Y. Zhang, and T. Okatani, "Visualization of convolutional neural networks for monocular depth estimation," in *Proc. IEEE/CVF Int. Conf. Comput. Vis. (ICCV)*, Oct. 2019, pp. 3869–3878.

[34] K. Xian, C. Shen, Z. Cao, H. Lu, Y. Xiao, R. Li, and Z. Luo, "Monocular relative depth perception with Web stereo data supervision," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit.*, Jun. 2018, pp. 311–320.

[35] C. Cadena, A. R. Dick, and I. D. Reid, "Multi-modal auto-encoders as joint estimators for robotics scene understanding," in *Proc. Robot. Sci. Syst.*, AnnArbor, MI, USA, Jun. 2016, pp. 1–9.

[36] T.-H. Wang, F.-E. Wang, J.-T. Lin, Y.-H. Tsai, W.-C. Chiu, and M. Sun, "Plug-and-play: Improve depth prediction via sparse data propagation," in *Proc. Int. Conf. Robot. Automat. (ICRA)*, May 2019, pp. 5880–5886.

[37] Y. Li, K. Qian, T. Huang, and J. Zhou, "Depth estimation from monocular image and coarse depth points based on conditional GAN," in *Proc. MATEC Web Conf.*, vol. 175, 2018, p. 03055.

[38] Z. Huang, J. Fan, S. Cheng, S. Yi, X. Wang, and H. Li, "HMS-Net: Hierarchical multi-scale sparsity-invariant network for sparse depth completion," *IEEE Trans. Image Process.*, vol. 29, pp. 3429–3441, 2020.

[39] X. Yang, J. Chen, Z. Wang, Q. Zhang, W. Liu, C. Liao, and K.-T. Cheng, "Monocular camera based real-time dense mapping using generative adversarial network," in *Proc. 26th ACM Int. Conf. Multimedia*, 2018, pp. 896–904.

[40] M. Calonder, V. Lepetit, C. Strecha, and P. Fua, "BRIEF: Binary robust independent elementary features," in *Proc. Eur. Conf. Comput. Vis.*, 2010, pp. 778–792.

[41] R. Kümmerle, G. Grisetti, H. Strasdat, K. Konolige, and W. Burgard, "G^2o: A general framework for graph optimization," in *Proc. IEEE Int. Conf. Robot. Automat.*, May 2011, pp. 3607–3613.

[42] J. J. Moré, "The Levenberg-Marquardt algorithm: Implementation and theory," in *Numerical Analysis*. Berlin, Germany: Springer, 1978, pp. 105–116.

[43] D. Xu, W. Ouyang, X. Wang, and N. Sebe, "PAD-Net: Multi-tasks guided prediction-and-distillation network for simultaneous depth estimation and scene parsing," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, Jun. 2018, pp. 675–684.

[44] G. Xie, G. Zeng, R. Kurachi, H. Takada, Z. Li, R. Li, and K. Li, "WCRT analysis and evaluation for sporadic message-processing tasks in multicore automotive gateways," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 38, no. 2, pp. 281–294, Feb. 2019.

[45] K. Li, "Optimal power and performance management for heterogeneous and arbitrary cloud servers," *IEEE Access*, vol. 7, pp. 5071–5084, 2019.

XIAOHAN TU received the M.Sc. degree in computer science and technology from Hunan University, Changsha, China, in 2017, where she is pursuing the Ph.D. degree from the Key Laboratory for Embedded and Network Computing of Hunan Province. She is also participating with the Project of National Natural Science Foundation of China, namely, CPS Instantiation-Research on the Smart Inspection Robot of Catenary. She has presented research articles in the IEEE international conferences. Her research interests include cyber-physical systems, computer vision, and machine learning. She is also a Reviewer of IEEE Access.

CHENG XU received the Ph.D. degree in computer science and engineering from the Wuhan University of Technology, Wuhan, China, in 2006.

He is currently a Professor of computer science and electronic engineering with Hunan University, Changsha, China. He is also a Ph.D. Supervisor with the College of Information Science and Engineering, Hunan University. He has presented 28 articles and has hosted several national and provincial nature fund projects. His current research interests include cyber-physical systems, embedded systems, digital video processing, computer vision, and machine learning.

Dr. Xu is a member of China Computer Federation.

SIPING LIU received the M.Sc. degree in computer science and technology from Hunan University, Changsha, China, in 2017, where he is currently pursuing the Ph.D. degree in computer science and technology with the Key Laboratory for Embedded and Network Computing of Hunan Province. His research interests include embedded systems, parallel computing, cyber-physical systems, and machine learning.

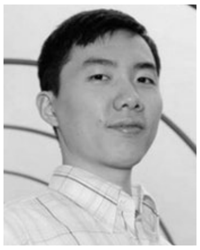
GUOQI XIE (Senior Member, IEEE) received the Ph.D. degree in computer science and engineering from Hunan University, China, in 2014.

He was a Postdoctoral Research Fellow with Nagoya University, Japan, from 2014 to 2015. He has been an Associate Professor with the Department of Computer Engineering, College of Computer Science and Electronic Engineering, Hunan University, since 2017. His current research interests include embedded and cyber-physical systems, parallel and distributed systems, and software engineering and methodology. He received the Best Paper Award at the IEEE ISPA 2016 and the 2018 IEEE TCSC Award for Excellence (Early Career Researcher). He is also serving on the Editorial Board of the *Journal of Systems Architecture*, the *Journal of Circuits, Systems, and Computers*, *Microprocessors and Microsystems*, and IEEE Access. He is an ACM Senior Member.

JING HUANG received the Ph.D. degree in computer science and technology from Hunan University, Changsha, China, in 2018. He is currently working as a Postdoctoral Researcher with Hunan University. His research interests include parallel computing, high-performance computing, distributed computing, energy-efficient computing, heterogeneous computing, cloud computing, and machine learning.

RENFA LI (Senior Member, IEEE) is currently a Professor of computer science and electronic engineering with Hunan University, Changsha, China. He is also the Director of the Key Laboratory for Embedded and Network Computing of Hunan Province, China. His current research interests include computer architectures, embedded computing systems, cyber-physical systems, and the Internet of Things.

Dr. Li is a member of the Council of China Computer Federation and a Senior Member of the Information Processing Society of Japan.



JUNSONG YUAN (Senior Member, IEEE) received the B.Eng. degree from the Special Program for the Gifted Young, Huazhong University of Science and Technology (HUST), China, the M.Eng. degree from the National University of Singapore, and the Ph.D. degree from Northwestern University.

He is currently an Associate Professor and the Director of Visual Computing Laboratory, Department of Computer Science and Engineering (CSE), State University of New York at Buffalo, USA. His research interests include computer vision, pattern recognition, video analytics, gesture and action analysis, large-scale visual search, and mining.

Dr. Yuan received 2016 Best Paper Award from the IEEE Transactions on Multimedia, the Doctoral Spotlight Award from the IEEE Conference on Computer Vision and Pattern Recognition (CVPR'09), a Nanyang Assistant Professorship from NTU, and the Outstanding EECS Ph.D. Thesis Award from Northwestern University. He is the Program Co-Chair of ICME18 and VCIP15 and the Area Chair of ACM MM18, ACCV1814, CVPR17, and ICIP17. He is also a Senior Area Editor of the *Journal of Visual Communication and Image Representation* (JVCI) and an Associate Editor of the IEEE Transactions on Image Processing (T-IP) and the IEEE Transactions on Circuits and Systems for Video Technology (T-CSVT). He has served as a Guest Editor of the *International Journal of Computer Vision* (IJCV).