Learning Meshes for Dense Visual SLAM

Michael Bloesch[1], Tristan Laidlow[1], Ronald Clark[1], Stefan Leutenegger[2], Andrew J. Davison[1]
[1]Dyson Robotics Laboratory at Imperial College [2] Smart Robotics Lab
Imperial College London, UK

Abstract

Estimating motion and surrounding geometry of a moving camera remains a challenging inference problem. From an information theoretic point of view, estimates should get better as more information is included, such as is done in dense SLAM, but this is strongly dependent on the validity of the underlying models. In the present paper, we use triangular meshes as both compact and dense geometry representation. To allow for simple and fast usage, we propose a view-based formulation for which we predict the in-plane vertex coordinates directly from images and then employ the remaining vertex depth components as free variables. Flexible and continuous integration of information is achieved through the use of a residual based inference technique. This so-called factor graph encodes all information as mapping from free variables to residuals, the squared sum of which is minimised during inference. We propose the use of different types of learnable residuals, which are trained end-to-end to increase their suitability as information bearing models and to enable accurate and reliable estimation. Detailed evaluation of all components is provided on both synthetic and real data which confirms the practicability of the presented approach.

1. Introduction

While a large amount of the interest in dense visual simultaneous localisation and mapping (SLAM) algorithms may be due to their ability to create fully dense 3D reconstructions of the environment (an important requirement for many tasks such as manipulator grasp planning, augmented reality, and safe robotic navigation), one of the original motivations behind dense SLAM was to attain higher levels of accuracy and robustness in localisation [23]. It was argued that since the quality of an estimate can only improve with additional measurements, dense approaches, which make use of all of the pixels in the image, should be capable of better performance than sparse systems.

This information theoretic claim, however, is only true under the assumption that the correct probabilistic model is used for all of the additional data. This has usually not been the case in practical dense SLAM systems. One key reason is that the direct photometric measurements that fully dense systems use, which depend on lighting and reflection, have residuals which are not well modelled in many parts of a scene by the same computationally simple Gaussian/M-estimator models which are suitable for sparse feature reprojection error. Further, just the large amount of measurement data from all-pixels measurements has meant that fully probabilistic inference which takes account of all cross correlations between estimates has been approximated by alternating pose/map optimisation. There is good discussion of these issues in [9, 24]. The outcome is that sparse feature-based SLAM with dense reconstruction as a layer on top has become the standard practical approach, rather than the appealing promise from [23] of using the dense model itself as the master map.

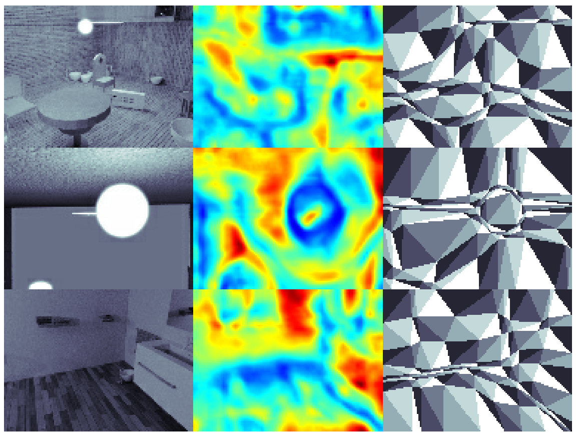
Figure 1: Input images, summed feature activations, and predicted triangular meshes for three example frames. While the in-plane vertex coordinates are directly predicted from the image, the remaining vertex depth coordinate is optimisable and will be inferred based on learned residuals.

One of the key factors determining both the computational cost and model accuracy of a dense inference algorithm is the choice of geometry representation. As in

CodeSLAM [1], we are interested in a representation that is suitable for keyframe-based dense visual SLAM; specifically, we require a representation that is powerful enough to capture the observed scene accurately, allows for information fusion, and is computationally efficient. For this reason, we choose to rely on triangular meshes as they are capable of representing dense geometry with few degrees of freedom. Similar to [38, 41], we model the meshes in 2.5D for numerical and computational considerations. In particular, we employ a view-based parameterisation that treats the image plane coordinates as *learnable* quantities and the vertex depth as an *optimisable* quantity. Specifically, a neural network predicts the image plane coordinates of the mesh vertices based on a single input image such that the scene geometry can be well reconstructed when optimising depth.

The choice of representation, however, is only one part of a full system. Measurements need to be continuously integrated to guarantee a good exploitation of the available information. In contrast to systems that learn to predict state updates directly [4], we desire a more flexible approach that allows for the continuous integration of novel information without being constrained to a fixed number of frames or optimisation steps. To this end, we use a factor graph formulation, a well-established methodology in the field of Bayesian estimation [16]. All available information is encoded in the form of residuals that are functions of the variables to be estimated (for example, in our case, the vertex depths for the triangular mesh). While this can be complemented with further residuals, we implement two types of residuals: a *prior* residual that holds prior information (e.g. smoothness) and a *stereo* residual that extracts information from different observations of the same scene content. Differing from traditional factor graphs, however, we propose learning the residuals within the optimisation framework so as to promote synergy between the various components. While this relies on the availability of large datasets, it enables the use of high dimensional and strongly descriptive models with performance beyond hand-tuned models.

In comparison with previous work and in particular CodeSLAM [1], which also advocates for the use of a compact representation to achieve dense visual SLAM, we introduce the following two key novelties:

- 2.5D triangular meshes as a compact representation for scene geometry where the image plane coordinates are predicted by a network from a single image and where the depth coordinates are optimisable quantities.

- A factor graph formulation that uses learnable residuals tailored to the proposed representation and allows the continuous and flexible integration of sensor information.

2. Related Work

Traditional dense monocular SLAM systems such as DTAM [23] typically represent scene geometry using keyframes with dense 2.5D depth maps. Estimating the per-pixel depth values is computationally expensive and therefore approximations to the inference method are made, usually alternately optimising for the pose and map and ignoring cross-correlations. The residuals used in these systems are based on the photometric error, which is not well-constrained in general due to homogeneous texture, occlusions, or other disturbances. To account for this, a regulariser is typically added, often based on smoothness [23] or planar [5] assumptions.

Recently, in order to overcome some of these issues with dense SLAM, a number of learning-based methods have been proposed. Many approaches (for example, [40, 32, 33, 37, 3]) advocate for a completely end-to-end system to predict pose and depth from video. While these systems typically perform well on autonomous driving datasets, the number of frames used in the estimation is fixed and there are often no guarantees of temporal consistency between predictions. Other approaches, notably CNN-SLAM [31], attempt to combine network predictions with stereo constraints. The stereo constraints are applied on a per-pixel basis, so global consistency is not preserved, and while CNN-SLAM performs a pose graph optimisation with each new keyframe, the scene geometry described in past keyframes is not optimised with new measurements. DeepTAM [39] uses separate tracking and mapping networks, the mapping network taking in a DTAM-style cost volume and then refining the prediction with several learning-based modules. Like traditional dense SLAM systems, DeepTAM does not jointly optimise for the pose and depth and the geometry of past keyframes are not updated with new information.

Due to the high computational cost of optimising fully dense scene geometry, there have been a number of attempts to find compact representations for dense 3D reconstruction. Most work has been based on exploiting simple regularities such as planar regions in indoor scenes [27, 15], but such hand-designed representations will ultimately be limited in the scenes they can accurately capture. Recently, CodeSLAM [1], the work most closely related to our own, proposed learning an optimisable code in the latent space of a depth auto-encoder, enabling the joint estimation of the pose and map in dense visual odometry. Similarly, BA-Net [30] uses a deep neural network to predict a set of basis depth maps using the coefficients of a linear combination of the basis elements as an optimisable representation for keyframe depth. They take a step further and train the representation in combination with a Levenberg-Marquardt (LM) optimisation with a fixed number of iterations and learnable damping factor. Unlike our system, BA-Net is not capable of jointly optimising several keyframes.

Meshes are a powerful and popular method for representing scene geometry [38]. There are numerous examples of methods that adapt meshes to image data (e.g. [12, 11]) or fit meshes to given points, depths, or disparities (e.g. [2, 14, 26, 10]). Other work attempts to generate 3D meshes using deep neural networks ([17, 34, 13]), optimisation ([7, 8]) or both [20], but these are typically focused on single object reconstruction. In MeshStereo [38], the authors emphasise the importance of obtaining meshes designed for the task at hand (in our case, achieving the best possible depth reconstruction). For this reason, we split the mesh vertex coordinates into learnable in-plane coordinates and optimisable depth coordinates and train a network to produce in-plane coordinates that minimise the reconstruction error after optimisation with ground truth.

Recently a number of data-driven alternatives have been proposed to the classical residuals used in dense visual SLAM. In [28] and [35], learning-based methods are proposed for developing robust features for dense image alignment. In [6], the authors experimented with using a "semantic texture" by using pre-trained CNN features for alignment. In addition to dense image alignment, other work has focused on using learning-based priors to constrain the optimisation of dense depth maps. For example, [36] uses network-predicted normals as a prior for dense 3D reconstruction. Our approach to learning residuals is novel in that the training is done directly in the optimisation framework ensuring that they are optimised for the purpose of dense visual SLAM using our mesh representation.

3. Image Conditioned Triangular Meshing

The relationship between representation and measurements plays a key role in any inference algorithm. While meshes can represent 3D scene structure with a small number of parameters, relating meshes to camera information typically involves some sort of rendering. This rendering process may be complex [18] and, if differential quantities such as Jacobians are required, too slow for real-time dense visual SLAM. To address these issues, we choose to use view-based 2.5D triangular meshes; that is, we express the vertex coordinates with respect to the camera frame and treat the in-plane and depth coordinates differently. The in-plane coordinates are treated as learnable quantities and are predicted by a deep neural network that conditions these coordinates on the image content. The depth coordinates of the vertices, however, represent the actual degree of freedom and are *not* predicted by any neural network, but are obtained through an optimisation procedure.

3.1. Mesh Rendering in 2.5D

To ensure differentiability, we use a fixed connectivity for the meshes and only alter the vertex coordinate $\boldsymbol{v}_k$ for every vertex $k \in \{1, \dots N\}$. The vertex coordinate is split into the in-plane coordinates $\boldsymbol{c}_k$ (in homogeneous representation) and inverse depth x_k, such that $\boldsymbol{v}_k = \boldsymbol{c}_k / x_k$. The motivation for using inverse depth is twofold: firstly, the uncertainty associated with inverse depths more closely follows a Gaussian distribution [22], and secondly, the relationship between the full inverse depth map and the inverse depth of the individual vertices is linear when assuming the triangles to be planar in 3D. This means that we can directly retrieve the inverse depth map D from the vertex inverse depths $\boldsymbol{x} = (x_0, \dots, x_N)$ through a linear map:

$$D(\boldsymbol{x}) = \boldsymbol{J}(\boldsymbol{c})\, \boldsymbol{x}, \tag{1}$$

where the map (or Jacobian) $\boldsymbol{J}(\boldsymbol{c})$ is a function of the in-plane coordinates $\boldsymbol{c} = (\boldsymbol{c}_0, \dots, \boldsymbol{c}_N)$ and is essentially composed of the barycentric weights that can be obtained via rasterisation.

Since the in-plane coordinates are constant at inference time, the association between pixels and triangles does not change and the above Jacobian $\boldsymbol{J}(\boldsymbol{c})$ can be precomputed for every frame after predicting the in-plane mesh coordinates. Furthermore, and in contrast to CodeSLAM [1], the Jacobian is a sparse matrix since the inverse depth of each pixel only depends on three relevant vertices. Sparse operations can therefore be used to reduce the computational load at both training and test time.

3.2. Predicting In-Plane Coordinates

The in-plane vertex coordinates $\boldsymbol{c}$ are predicted based on the image content I of the frame, i.e. $\boldsymbol{c} = \boldsymbol{c}(I)$. This is achieved using a deep neural network that first computes pixel-wise features using a U-Net [25] and then distorts a regular triangular mesh based on the feature activations in the local neighbourhood of the vertices (see Figure 2). The regular mesh is created by defining N_r equidistant vertex rows and then alternately distributing N_c and $N_c - 1$ vertices along these rows. Similar to the feature encoder network in CodeSLAM [1], the U-Net is composed of four pairs of down-sampling convolutions and four pairs of up-sampling convolutions with corresponding skip layers. The first convolutional layer in each pair uses a stride of 2 and the size of the feature channels are (16, 16, 32, 32, 64, 64, 128, 128) for the encoder, with analogous settings for the decoder. All convolutions have a kernel size of 3 (except the first, which has a size of 7) and are followed by Rectified Linear Units.

In order to go from feature activations to vertex coordinates, we extract patches around each of the regular mesh vertices and compute perturbations to the coordinate locations. Patches of size 25×25 are extracted from all 16 channels of the last layer and are directly transformed to vertex locations by passing each patch through a single fully-connected layer and adding its output to the coordinates of the regular mesh vertex. Border vertices are forced to stay on the frame boundary.

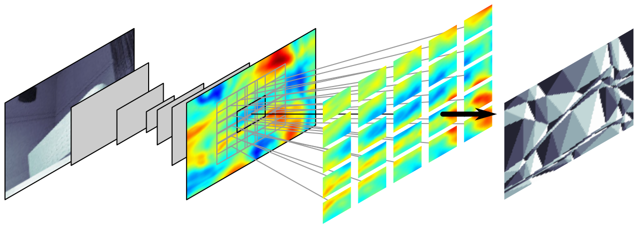

Figure 2: From image to mesh: input images are fed through a U-Net with skip layers to produce pixel-wise features. Patches at regular mesh vertices are extracted and then passed through a fully-connected layer to produce perturbations to the vertex locations.

3.3. Training Loss

The goal of the training process is to obtain a network capable of generating meshes suitable for representing view-based 3D geometry. We propose to do this by defining a training loss directly on the 3D reconstruction. For a given set of in-plane vertex coordinates, we compute the corresponding vertex depths that best fit the inverse depth map in a least squares sense. Since the relationship between the inverse depth coordinates of the vertices $\boldsymbol{x}$ and inverse depth map $D(\boldsymbol{x})$ is linear, we can compute the best fit in a single step by solving the normal equation:

$$\boldsymbol{J}(\boldsymbol{c})^T \boldsymbol{J}(\boldsymbol{c})\, \boldsymbol{x} = \boldsymbol{J}(\boldsymbol{c})^T D. \tag{2}$$

As we rely on a triangular mesh to approximate the environment and, in general, this will never result in a perfect fit, there will be some remaining reconstruction error (see the examples in Figure 3). This can be computed by solving for $\boldsymbol{x}$ and then substituting in eq. (1):

$$E = \left(\boldsymbol{I} - \boldsymbol{J}(\boldsymbol{c})(\boldsymbol{J}(\boldsymbol{c})^T \boldsymbol{J}(\boldsymbol{c}))^{-1} \boldsymbol{J}(\boldsymbol{c})^T\right) D. \tag{3}$$

After eliminating $\boldsymbol{x}$, the final reconstruction error will be a function of the predicted in-plane coordinates $\boldsymbol{c}$. Thus, using the reconstruction error as loss, coordinates that result in lower reconstruction errors will be encouraged, which is typically achieved by having the coordinates coincide with regions of high curvature (see Figure 1).

The final loss is defined to be the squared residual error after optimisation, $\|E\|^2$, and this can be implemented, including back-propagation, in most state-of-the-art deep learning frameworks. The computation of $\boldsymbol{J}(\boldsymbol{c})$ relies on differential rasterisation and, to allow for fast implementation, we assume that the mesh triangles cannot exceed a certain size (half of the image height) and that there are no overlapping triangles. This significantly simplifies and accelerates the computations and these two extra assumptions are enforced through auxiliary losses. When comparing to CodeSLAM [1], the inverse depth rendering takes over the functionality of the decoder and this is conditioned on the image content via the predicted in-plane coordinates $\boldsymbol{c}$. Since we do not require an encoder (as this functionality is now performed by the least squares optimisation), we have the advantage that the network can be trained with incomplete depth data where missing pixels are ignored.

4. Learning of Error Metrics and Priors

In a monocular vision setup, both the geometry of the environment and the motion of the camera are unknown. Similar to other recent work [1, 30], we want to jointly estimate both unknown quantities in a keyframe-based setup. This problem can be formulated as a factor graph [16] where each keyframe i is associated with two variables: the pose T_i and the inverse depth of the vertices $\boldsymbol{x}_i$. Any information concerning parts of the graph is then encoded by means of factors that are connected to the relevant variables. In the present setup, two types of factors are proposed:

- A *prior* factor that is only connected to the inverse depths of a single keyframe. This can be used to encode a smoothness constraint and reduce curvature of the mesh, for example.
- A *stereo* factor that is connected to the poses and inverse depths of two keyframes. This is used to combine the information from the images with the knowledge that both are observations of the same scene.

Employing a residual notation, the prior factor is encoded as a mapping $p_i(\boldsymbol{x}_i)$ and the stereo factor is encoded as a mapping $s_{ij}(\boldsymbol{x}_i, \boldsymbol{x}_j, T_i, T_j)$ where i and j are keyframe indices and where the mappings may depend on the images I_i and I_j. The advantages of this approach are that the factors can be combined to represent an arbitrary number of frames, that additional information can be easily encoded and added to the system (for example, from other sensor modalities), and that other probabilistic methodologies such as marginalisation (variable elimination) can be applied.

4.1. Prior Factor

The prior factor $p_i(\boldsymbol{x}_i)$ explicitly encodes prior knowledge on the scene geometry and may be conditioned on the image I_i – CodeSLAM uses a similar concept via the KL-loss on the latent space. But the prior could also be modelled manually and for instance penalise rough meshes which are unlikely to correspond to real scenes. However, here we propose to learn a data-driven prior by using a similar setup as was employed in Section 3.2 to predict the in-plane vertex coordinates. In particular, we use the same U-net architecture to predict feature activations and extract per-vertex patches. Based on these patches three different types of linear sparse priors are computed. In order to encode independent vertex depth information we directly

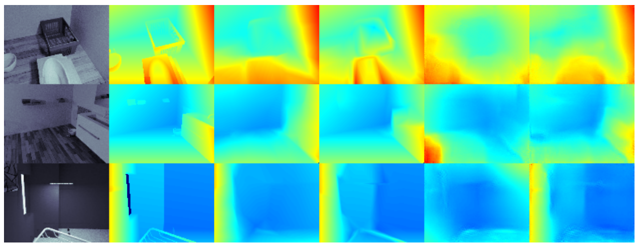

Figure 3: Comparison of inverse depth reconstructions based on different representations. From left to right: input image, ground-truth inverse depth, mesh-based with learned prior, mesh-based optimised against ground truth, CodeSLAM-based with prior, CodeSLAM-based optimised against ground truth.

transform the per-vertex patch P_k to a per-vertex residual $p_{\text{vtx}}(x_k) = a_{\text{vtx}}(P_k)\, x_k + b_{\text{vtx}}(P_k)$ where a_{vtx} and b_{vtx} are implemented as fully connected layers. This is extended to allow triangle-related information to be encoded as well, such as inclination cues. To this end, for every triangle $\boldsymbol{t} = (x_k, x_l, x_m)$ with stacked patch $P_t = (P_k, P_l, P_m)$ a linear residual of the form $p_{tri}(\boldsymbol{t}) = a_{tri}(P_t)\, \boldsymbol{t} + b_{tri}(P_t)$ is computed. Finally, the same is done for edges $\boldsymbol{e} = (x_k, x_l, x_m, x_n)$ which includes all vertices of the two adjacent triangles. All components are aggregated into a sparse matrix $A(I)$ and a vector $b(I)$ to obtain a combined linear prior on all vertex inverse depths:

$$p(\boldsymbol{x}) = A(I)\, \boldsymbol{x} + b(I). \tag{4}$$

Figure 3 provides a visualisation of this prior by depicting its least squares solution.

4.2. Stereo Factor

The stereo factor is used to exploit overlapping fields of view between keyframes. It models the information from the images when combined with the fact that they represent observations of the same scene. The corresponding residual $s_{ij}(\boldsymbol{x}_i, \boldsymbol{x}_j, T_i, T_j)$ may and should depend on the images I_i and I_j. A possible example for such a residual would be the photometric error. This can be implemented by first computing the dense correspondences for all pixels from frame i to frame j

$$\boldsymbol{u}_j = \pi(T_j^{-1} T_i \pi^{-1}(\boldsymbol{u_i}, D(\boldsymbol{x}_i)[\boldsymbol{u_i}]), \tag{5}$$

where the square bracket notation $[\boldsymbol{u}]$ stands for the lookup at pixel $\boldsymbol{u}$, π projects homogeneous point coordinates to pixel coordinates, and π^{-1} back-projects to homogeneous point coordinates using the inverse depth. With this, the standard uni-directional pixel-wise photometric residual can be formulated as [19]:

$$s_{iju}(\boldsymbol{x}_i, \boldsymbol{x}_j, T_i, T_j) = I_i[\boldsymbol{u}_i] - I_j[\boldsymbol{u}_j]. \tag{6}$$

While in perfectly Lambertian and smooth environments the use of the photometric residual is appropriate, it often has a small convergence basin and lacks robustness with respect to non-Lambertian effects and occlusions. Thus we want to go a step further and attempt to learn the stereo residual based on its alignment performance. As a first design specification we want the residual to be scale independent and thus choose it to be a function of the dense correspondences $\boldsymbol{u}_i, \boldsymbol{u}_j$ only. Additionally, we want the residual to be a fairly simple function of the correspondences in order to enable pre-computation and allow fast iterations. Therefore, we propose to first pre-process both images i and j by $Y(\cdot)$, then lookup the correspondences, and finally apply some lightweight mapping $r(\cdot)$:

$$s_{iju}(\boldsymbol{x}_i, \boldsymbol{x}_j, T_i, T_j) = r(Y(I_i)[\boldsymbol{u}_i], Y(I_j)[\boldsymbol{u}_j]). \tag{7}$$

This is a slightly more general form of the feature-metric error [6, 30] and has the ability to revert to photometric error if this is the best solution. We again exploit convolutions and choose $Y(\cdot)$ to have the same architecture as discussed in Section 3.2, except for a different number of output feature channels N_y. After predicting features in both images and matching them via dense correspondences, the mapping $r(y_i, y_j)$ generates a residual for every pixel. It is composed of two parts (see also Figure 4):

$$r(y_i, y_j) = (y_i^n - y_j^n) \cdot w(y_i, y_j), \tag{8}$$

where the first part computes the difference of the n-th elements of the feature vectors and does not include any additional learned components to avoid redundancy. The second part is composed of a multilayer perceptron (MLP) $w(\cdot)$ that generates a weighting for the residual. It is composed of 3 hidden layers of size N_w and has a scalar output. We experimented with different combinations of feature channels N_y, residual dimensions, and hidden layer dimension N_w and found that a combination of $N_y = 3$, a single residual, and $N_w = 3$ performed well. While during optimisation no gradients are computed through w for efficiency reasons, w can take over a weighting functionality and for instance down-weight unreliable correspondences. Please note that the back-propagation during training will be computed through the optimisation steps and will update all learnable components including the MLP w.

4.3. Training Setup

It is often beneficial to keep the training as close as possible to the test setup, and thus we train the network weights by rolling out the internal factor graph optimisation, which is done via damped Gauss-Newton iterations. For computational reasons, however, both the number of variables as well as optimisation steps must be limited (see Figure 5). We only optimise the inverse vertex depths for a

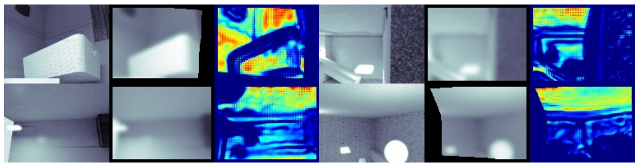

Figure 4: Input image together with generated textures and weightings (blue low, red high). Both are learned as part of the stereo residual r. On the lower left example one can observe how the specularity on the wall is down-weighted.

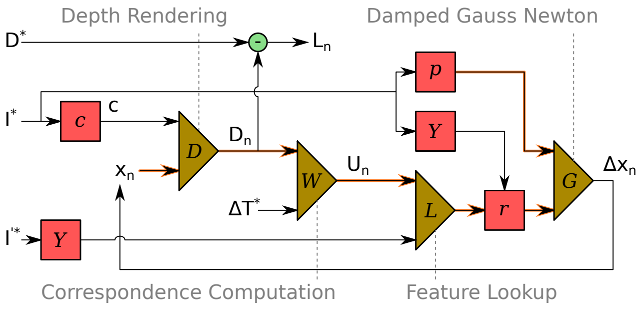

Figure 5: Setup for training the learnable residuals based on paired frames. The initial depth vertex coordinates $\boldsymbol{x}_0$ are initialised to a constant value and combined with the mesh network output $\boldsymbol{c}$ to generate a depth map D. This is subsequently transformed into dense correspondences U using ground truth relative pose ΔT^*. Finally, both the prior p and stereo residual r are evaluated and used to compute the damped Gauss-Newton update step $\Delta \boldsymbol{x}$. Since this requires computation of Jacobians, these are also computed (highlighted arrows). During training, multiple steps are rolled out and the reconstruction error is used to update the residual-related weights in p, Y, and r.

uni-directional two-frame setup and take 3 iterations steps only. That is, during training, we sample image pairs I_i and I_j together with their camera poses T_i and T_j from the dataset and construct the least squares problem as follows:

$$\begin{aligned} L(\boldsymbol{x}_i) &= \|p(\boldsymbol{x}_i)\|^2 + \sum_{\boldsymbol{u}_i} s^2_{iju}(\boldsymbol{u}_i, \boldsymbol{u}_j) \qquad (9) \\ &= \|A(I_i)\, \boldsymbol{x}_i + b(I_i)\|^2 + \sum_{\boldsymbol{u}_i} r^2(Y_i[\boldsymbol{u}_i], Y_j[\boldsymbol{u}_j]), \end{aligned}$$

where the terms are written out using the abbreviation $Y_i = Y(I_i)$. While $\boldsymbol{x}_i$ is the variable that is affected by the optimisation steps, the training will backpropagate the final loss to the learnable prior components A and b and stereo components Y and w (via r).

The initialisation of the inverse depth plays an important role. After observing that the network would exploit any form of noisy initialisation procedure, we chose to initialise the inverse depth to a constant value. The final loss for training is composed of the inverse depth reconstruction loss on the full image and uses an L1 norm to avoid disturbing effects from very near objects (which lead to very high losses). Furthermore, rather then just using the reconstruction error after 3 iteration steps, the errors are summed over all iterations and weighted by the error of the previous iteration. This again down-weights training examples with very near objects which may disturb the training procedure.

In summary, the differences to the test setup are the limited number of iterations, the lack of pose optimisation, the constant initialisation of the inverse depth, and the use of two keyframes only. The following evaluation section will investigate some of these discrepancies. However, we would like to highlight that in comparison to many other approaches, the use of a residual-based formulation offers a high degree of modularity and among other benefits enables the combination of an arbitrary number of keyframes.

5. Experimental Evaluation and Discussion

5.1. Datasets and Training Setup

During training we require paired frames with overlapping fields of view. In terms of data modality, we need access to images, depths, and poses of the associated frames. We use SceneNet RGB-D [21] (SN) as a synthetic dataset and TUM's RGB-D SLAM dataset (TM) for real data [29]. For both, we transform the images to grayscale with a resolution of 128×96 and create pairs by picking consecutive frames (we skip 7 frames for TM to increase the baseline). We found it necessary to augment the dataset by perturbing the overall scale in order to avoid over-fitting the learned prior. This step was particularly important when fine-tuning on TM due to the limited dataset size.

We use $N_r = 9$ rows of vertices and $N_c = 11$ columns (95 vertices). All networks are trained using the ADAM optimiser. The mesh network is trained for 8 epochs on SN using a batch size of 64 and an initial learning rate of $1e-4$. It is fine-tuned on TM during 30 epochs with a learning rate of 1e-5. The residual generating network is trained for 6 epochs on SN using a batch size of 32 and an initial learning rate of 1e-4. It is fine-tuned on TM for 40 epochs.

5.2. Meshes for Scene Depth

Qualitative examples of the predicted meshes are shown in Figure 1. We observe how the vertices of the meshes are attracted towards depth discontinuities or regions of high curvature such as room corners. The figure also depicts the summed activations from the last layer of the U-net that occurs before the patch extraction. Blue regions (low activations) attract vertices while red regions repel them.

Different properties can be evaluated to analyse the suitability of the proposed mesh representation for visual mo-

Table 1: Comparison of the ability of different sparse representations to produce dense depth maps. Learned mesh trained on SN (Mesh SN), learned mesh trained on SN and fine-tuned on TM (Mesh TM), a regular mesh with the same number of vertices as the learned mesh, and CodeSLAM are evaluated on the L1-inv error described in [39] on both the SN and TM datasets. Each representation is optimised against the ground truth depth map.

DS	Error	Mesh SN	Mesh TM	Reg. Mesh	CodeSLAM
SN	L1_inv	**0.0091**	0.0105	0.0159	0.0198
	L1_rel	**0.0220**	0.0247	0.0370	0.0464
TM	L1_inv	0.0208	**0.0188**	0.0267	0.0346
	L1_rel	0.0462	**0.0420**	0.0586	0.0751

Table 2: Comparison of reconstruction performance when using learned priors. For metrics marked AS, images have been auto-scaled by a single factor to match ground truth.

DS	Error	Mesh SN	Mesh TM	CodeSLAM
SN	L1_inv	0.1864	0.2515	**0.1692**
	L1_rel	**0.3702**	0.5907	0.3742
	L1_inv_AS	**0.0744**	0.1250	0.1243
	L1_rel_AS	**0.1566**	0.2751	0.2814
TM	L1_inv	0.2897	**0.1291**	0.2981
	L1_rel	0.5786	**0.2289**	0.6056
	L1_inv_AS	0.1485	**0.0704**	0.1710
	L1_rel_AS	0.3053	**0.1427**	0.3675

tion and geometry estimation. We first look at the best possible reconstructions that can be obtained by fitting the mesh to the ground truth inverse depth map. This is done for both SN and TM and we compare the meshing trained on SN only, the meshing fine-tuned on TM, a regular meshing, and a 128D encoding based on CodeSLAM [1]. The same inverse L1 (L1_inv [1/m]) reconstruction metric as DeepTAM [39] and DeMoN [32] as well as a relative error (L1_rel [1]) are reported in Table 1. All reported values are averages over 1000 samples. For SN, these are the first examples of the 1000 validation sequences. For TM, every 40th frame is sampled. With an L1_inv of below 0.01 1/m for SN, the achievable reconstruction error is very low and shows that this is not a limitation of the proposed mesh representation. The error is also lower than those obtained with either a regular mesh or a CodeSLAM based encoding. While rather marginal, there is some benefit in fine-tuning the meshing on TM, but this is overall a difficult endeavour due to the small size of the dataset (77,555 training examples).

Figure 3 shows some example of optimised reconstructions against ground-truth for both meshing and CodeSLAM. It also depicts the learned priors, another set of quantities we want to investigate. While the prior on its own need not necessarily match a meaningful depth map, the obtained prior based predictions suggest that this is the case and that these could for instance be used for initialisation. We quantify this in Table 2, which reports the reconstruction accuracy of the priors on SN and TM for the learned mesh and CodeSLAM (via zero code prediction). As expected, the obtained errors are increased when compared to the optimal meshes from before. In comparison to the optimal meshes, a larger effect can be observed when fine-tuning on TM. Also CodeSLAM seems to achieve similar one-shot reconstructions, but this is explained by the stronger scale prior it encodes. After correcting for a scale offset, our mesh method leads to significantly lower errors.

Table 3: Two frame evaluation with different optimisation setups (*Fixed*: fixed pose, *Free*: free pose, *Free + Scale*: free pose with extra residual on mean inverse depth) and initialisation (*gt*: groundtruth, *gtn*: groundtruth with noise, *const*: constant, *id*: identity). All L1_inv.

Loss	DS	Depth	Pose	Fixed	Free	Free + Scale
Featuremetric	SN	gt	gt	0.0439	0.1343	0.0592
	SN	gtn	gt	0.0438	0.1342	0.0595
	SN	const	gt	0.0499	0.1688	0.0635
	SN	gt	gtn	–	0.1731	0.0674
	SN	gt	id	–	0.1616	0.0704
	SN	const	id	–	0.2082	0.0803
	TM	gt	id	–	0.1400	0.1373
	TM	const	id	–	0.2799	0.1437
Photometric	SN	gt	gt	0.0581	0.0614	0.0579
	SN	gtn	gt	0.0578	0.0639	0.0588
	SN	const	gt	0.1321	0.2017	0.1167
	SN	gt	gtn	–	0.1341	0.1159
	SN	gt	id	–	0.1552	0.1453
	SN	const	id	–	0.2842	0.2182

Table 4: Multi-frame results, L1_inv averaged over 16 SN samples.

Frames	1	2	4	8	16
L1_inv	0.0687	0.0509	0.0407	0.0339	0.0304

5.3. Multi-Frame Setup

The training procedure is distinct from the final setup in multiple ways. This includes the number of employed frames, the selection of variables, the number of iterations, and initialisation of the variables.

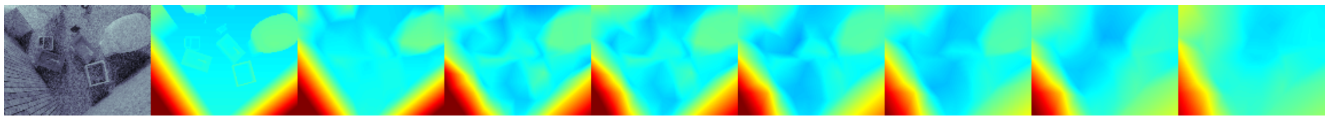

Figure 6: Single key-frame optimised against a varying number of frames.

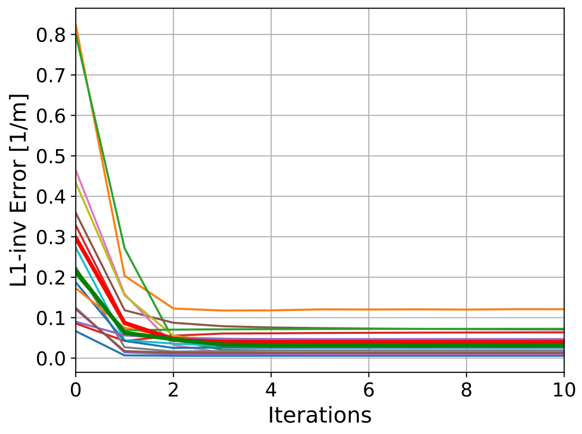

Figure 7: L1_inv reconstruction error over 10 iterations for 16 SN examples. Green: median, red: mean.

Table 3 collects multiple results from experiments done in a two-frame setup. In contrast to the training, this performs 20 Gauss-Newton iteration steps where the damping has been adapted if required. Furthermore it shows the final reconstruction error for different initialisations and different optimisation procedures. The later include optimisation with fixed pose (analogous to training), optimisation with free pose, and optimisation with free pose but with additional mean inverse depth constraint (implemented as additional residual using the ground-truth mean inverse depth). We can observe that when we open the pose degree of freedom, the reconstructions are worse. This is due to scale drift caused by the scale free training (the scale is randomly perturbed to improve balance between prior and stereo residual). This is confirmed by the last optimisation setup which introduces scale information via an extra residual and achieves similar result to the setup with fixed ground-truth pose. Furthermore, we provide results for different initialisations ranging from ground truth to constant depth and identity pose. For both SN and TM (no ground truth pose is available for validation), we observe that using the additional scale residual limits the effect of different initialisations. In particular, identity pose initialisation can be very far from the actual ground truth and this shows that we attain a large convergence basin.

In Table 3 we also provide results with a hand-engineered optimisation procedure that uses the predicted meshes but then employs photometric error and a curvature loss on the mesh. While this seems to achieve similar performance when good initialisation is provided, its performance deteriorates with poor initialisation. We can explain our increased convergence basin when looking at the feature activations used for the stereo residual and the corresponding weighting (see Figure 4). The visualised texture exhibits much higher smoothness compared to the original image, and when looking at the weighting one can observe that disturbances such as specularities are masked out.

Figure 7 shows the error over iteration steps for multiple examples. We can observe that the reconstruction error decreases consistently and does not degrade even if we go pass the three iterations used during training. Finally, in Figure 6 and Table 4 we report results where we optimise the depth of a keyframe against multiple other frames. With increasing number of frames the error decreases even though during training only single pairs were ever used.

6. Conclusion

In this paper we investigated the use of triangular meshes for joint estimation of motion and geometry using a monocular camera. We looked into how we can make the use of meshes efficient by using a view-based formulation, predicting in-plane vertex coordinates using a deep neural network, and keeping the vertex depths as optimisable quantities. In a second step, this was complemented by learned residuals to allow modular integration of the available information. This was used to learn both a purely data-driven prior as well as a residual leveraging the information contained in images resulting from observing the same scene.

Extensive experimental evaluations on both synthetic and real datasets confirmed many of the design choices, but also the importance of keeping the training close to the final setup as well as the key role of the data. Continuing from here, we want to investigate methods that reduce dataset requirements, such as for instance self-supervised loss functions. We also plan to scale to a fully-fledged SLAM system, possibly investigating world-centric meshes and the use of incremental editing and refinement procedures.

Acknowledgments

Research presented in this paper has been supported by Dyson Technology Ltd.

References

[1] Michael Bloesch, Jan Czarnowski, Ronald Clark, Stefan Leutenegger, and Andrew J. Davison. CodeSLAM — learning a compact, optimisable representation for dense visual SLAM. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2018. 2, 3, 4, 7

[2] András Bódis-Szomorú, Hayko Riemenschneider, and Luc Van Gool. Superpixel Meshes for Fast Edge-Preserving Surface Reconstruction. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2015. 3

[3] Vincent Casser, Soeren Pirk, Reza Mahjourian, and Anelia Angelova. Depth Prediction Without the Sensors: Leveraging Structure for Unsupervised Learning from Monocular Videos. In *Proceedings of the National Conference on Artificial Intelligence (AAAI)*, 2019. 2

[4] Ronald Clark, Michael Bloesch, Jan Czarnowski, Stefan Leutenegger, and Andrew J. Davison. LS-Net: Learning to Solve Nonlinear Least Squares for Monocular Stereo. In *Proceedings of the European Conference on Computer Vision (ECCV)*, 2018. 2

[5] Alejo Concha and Javier Civera. DPPTAM: Dense Piecewise Planar Tracking and Mapping from a monocular sequence. In *Proceedings of the IEEE/RSJ Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2015. 2

[6] Jan Czarnowski, Stefan Leutenegger, and Andrew J. Davison. Semantic texture for robust dense tracking. In *Proceedings of the International Conference on Computer Vision Workshops (ICCVW)*, 2017. 3, 5

[7] Amaël Delaunoy and Marc Pollefeys. Photometric Bundle Adjustment for Dense Multi-View 3D Modeling. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2014. 3

[8] Amaël Delaunoy and Emmanuel Prados. Gradient flows for optimizing triangular mesh-based surfaces: Applications to 3D reconstruction problems dealing with visibility. *International Journal of Computer Vision (IJCV)*, 95(2):100–123, 2011. 3

[9] Jakob Engel, Vladlen Koltun, and Daniel Cremers. Direct sparse odometry. *IEEE Transactions on Pattern Analysis and Machine Intelligence (PAMI)*, 2018. 1

[10] Guilherme Pinto Fickel, Claudio R. Jung, Tom Malzbender, Ramin Samadani, and Bruce Culbertson. Stereo Matching and View Interpolation based on Image Domain Triangulation. *IEEE Transactions on Image Processing*, 22:3353–3365, 2013. 3

[11] Orcun Goksel and Septimiu E. Salcudean. Image-Based Variational Meshing. *IEEE Transactions on Medical Imaging*, 30:11–21, 2011. 3

[12] Alberto Gomez, Veronika A. Zimmer, Bishesh Khanal, Nicolas Toussaint, and Julia A. Schnabel. Oversegmenting Graphs. *arXiv preprint arXiv:1806.00411*, 2018. 3

[13] Thibault Groueix, Matthew Fisher, Vladimir G. Kim, Bryan Russell, and Mathieu Aubry. AtlasNet: A Papier-Mâché Approach to Learning 3D Surface Generation. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2018. 3

[14] Rui Huang, Danping Zou, Richard Vaughan, and Ping Tan. Active image-based modeling with a toy drone. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, 2018. 3

[15] Michael Kaess. Simultaneous localization and mapping with infinite planes. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, 2015. 2

[16] Michael Kaess, Hordur Johannsson, Richard Roberts, Viorela Ila, John Leonard, and Frank Dellaert. iSAM2: Incremental Smoothing and Mapping Using the Bayes Tree. *International Journal of Robotics Research (IJRR)*, 2012. To appear. 2, 4

[17] Angjoo Kanazawa, Shubham Tulsiani, Alexei A. Efros, and Jitendra Malik. Learning Category-Specific Mesh Reconstruction from Image Collections. In *Proceedings of the European Conference on Computer Vision (ECCV)*, 2018. 3

[18] Hiroharu Kato, Yoshitaka Ushiku, and Tatsuya Harada. Neural 3D Mesh Renderer. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2018. 3

[19] Christian Kerl, Jürgen Sturm, and Daniel Cremers. Dense visual SLAM for RGB-D cameras. In *Proceedings of the IEEE/RSJ Conference on Intelligent Robots and Systems (IROS)*, 2013. 5

[20] Chen-Hsuan Lin, Oliver Wang, Bryan C Russell, Eli Shechtman, Vladimir G Kim, Matthew Fisher, and Simon Lucey. Photometric Mesh Optimization for Video-Aligned 3D Object Reconstruction. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2019. 3

[21] John McCormac, Ankur Handa, Stefan Leutenegger, and Andrew J. Davison. SceneNet RGB-D: Can 5M synthetic images beat generic ImageNet pre-training on indoor segmentation? In *Proceedings of the International Conference on Computer Vision (ICCV)*, 2017. 6

[22] José María Martinez Montiel, Javier Civera, and Andrew J. Davison. Unified Inverse Depth Parametrization for Monocular SLAM. In *Proceedings of Robotics: Science and Systems (RSS)*, 2006. 3

[23] Richard A. Newcombe, Steven Lovegrove, and Andrew J. Davison. DTAM: Dense Tracking and Mapping in Real-Time. In *Proceedings of the International Conference on Computer Vision (ICCV)*, 2011. 1, 2

[24] Lukas Platinsky, Andrew J. Davison, and Stefan Leutenegger. Monocular visual odometry: Sparse joint optimisation or dense alternation? In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, 2017. 1

[25] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. U-Net: Convolutional networks for biomedical image segmentation. In *Proceedings of the International Conference on Medical Image Computing and Computer Assisted Intervention (MICCAI)*, 2015. 3

[26] Antoni Rosinol, Torsten Sattler, Marc Pollefeys, and Luca Carlone. Incremental Visual-Inertial 3D Mesh Generation with Structural Regularities. *arXiv preprint arXiv:1903.01067*, 2019. 3

[27] Renato F. Salas-Moreno, Ben Glocker, Paul H. J. Kelly, and Andrew J. Davison. Dense planar SLAM. In *Proceedings of the International Symposium on Mixed and Augmented Reality (ISMAR)*, 2014. 2

[28] Tanner Schmidt, Richard Newcombe, and Dieter Fox. Self-supervised visual descriptor learning for dense correspondence. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, 2017. 3

[29] Jürgen Sturm, Nikolas Engelhard, Felix Endres, Wolfram Burgard, and Daniel Cremers. A Benchmark for the Evaluation of RGB-D SLAM Systems. In *Proceedings of the IEEE/RSJ Conference on Intelligent Robots and Systems (IROS)*, 2012. 6

[30] Chengzhou Tang and Ping Tan. BA-Net: Dense Bundle Adjustment Networks. In *Proceedings of the International Conference on Learning Representations (ICLR)*, 2019. 2, 4, 5

[31] Keisuke Tateno, Federico Tombari, and Nassir Navab. Real-time and scalable incremental segmentation on dense slam. In *Proceedings of the IEEE/RSJ Conference on Intelligent Robots and Systems (IROS)*, 2015. 2

[32] Benjamin Ummenhofer, Huizhong Zhou, Jonas Uhrig, Nikolaus Mayer, Eddy Ilg, Alexey Dosovitskiy, and Thomas Brox. DeMoN: Depth and motion network for learning monocular stereo. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2017. 2, 7

[33] Chaoyang Wang, Jose Miguel Buenaposada, Rui Zhu, and Simon Lucey. Learning Depth from Monocular Videos using Direct Methods. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2018. 2

[34] Nanyang Wang, Yinda Zhang, Zhuwen Li, Yanwei Fu, Wei Liu, and Yu-gang Jiang. Pixel2Mesh: Generating 3D Mesh Models from Single RGB Images. In *Proceedings of the European Conference on Computer Vision (ECCV)*, 2018. 3

[35] Chamara Saroj Weerasekera, Ravi Garg, Yasir Latif, and Ian Reid. Learning Deeply Supervised Good Features to Match for Dense Monocular Reconstruction. In *Proceedings of the Asian Conference on Computer Vision (ACCV)*, 2018. 3

[36] Chamara Saroj Weerasekera, Yasir Latif, Ravi Garg, and Ian Reid. Dense monocular reconstruction using surface normals. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, 2017. 3

[37] Huangying Zhan, Ravi Garg, Chamara Saroj Weerasekera, Kejie Li, Harsh Agarwal, and Ian Reid. Unsupervised Learning of Monocular Depth Estimation and Visual Odometry With Deep Feature Reconstruction. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2018. 2

[38] Chi Zhang, Zhiwei Li, Yanhua Cheng, Rui Cai, Hongyang Chao, and Yong Rui. MeshStereo: A Global Stereo Model with Mesh Alignment Regularization for View Interpolation. In *Proceedings of the International Conference on Computer Vision (ICCV)*, 2015. 2, 3

[39] Huizhong Zhou, Benjamin Ummenhofer, and Thomas Brox. DeepTAM: Deep tracking and mapping. In *Proceedings of the European Conference on Computer Vision (ECCV)*, 2018. 2, 7

[40] Tinghei Zhou, Matthew Brown, Noah Snavely, and David G. Lowe. Unsupervised learning of depth and ego-motion from video. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2017. 2

[41] Jacek Zienkiewicz, Akis Tsiotsios, Andrew J. Davison, and Stefan Leutenegger. Monocular, Real-Time Surface Reconstruction using Dynamic Level of Detail. In *Proceedings of the International Conference on 3D Vision (3DV)*, 2016. 2