DeepFactors: Real-Time Probabilistic Dense Monocular SLAM

Jan Czarnowski, Tristan Laidlow, Ronald Clark, and Andrew J. Davison

***Abstract*—The ability to estimate rich geometry and camera motion from monocular imagery is fundamental to future interactive robotics and augmented reality applications. Different approaches have been proposed that vary in scene geometry representation (sparse landmarks, dense maps), the consistency metric used for optimising the multi-view problem, and the use of learned priors. We present a SLAM system that unifies these methods in a probabilistic framework while still maintaining real-time performance. This is achieved through the use of a learned compact depth map representation and reformulating three different types of errors: photometric, reprojection and geometric, which we make use of within standard factor graph software. We evaluate our system on trajectory estimation and depth reconstruction on real-world sequences and present various examples of estimated dense geometry.**

***Index Terms*—Deep learning in robotics and automation, mapping, SLAM.**

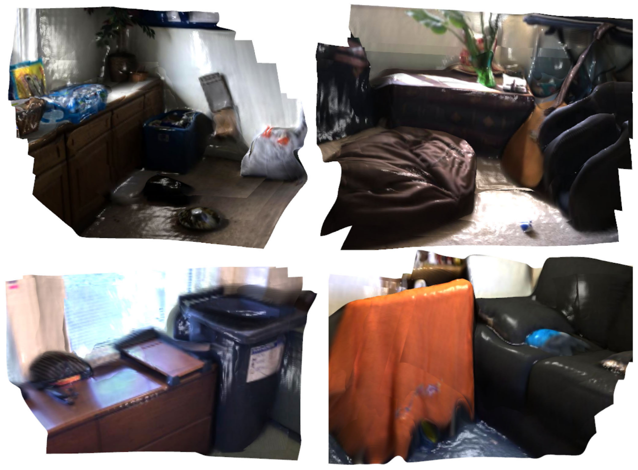

Fig. 1. Example reconstructions of scenes from the ScanNet validation set created with our system. Reflective shading has been exaggerated in order to highlight structure.

I. Introduction

Research in monocular SLAM has been divided into two paradigms that mainly differ in the geometric map representation. *Sparse* methods [1]–[3] form and maintain a map consisting of a set of point landmarks which are estimated through observing, identifying and tracking them in the camera images. In order to enable repeatable recognition and reliable matching of landmark observations, the map is limited to a relatively small number of salient points that are characteristic and easily distinguishable in their image projections (typically corners or edges). This limits the usefulness of resulting map reconstructions, which cannot be used in interactive robotics tasks or advanced augmented reality applications. At the same time, the size of the representation allows for real-time joint probabilistic inference. Since sparse methods rely on gradient-based keypoint detectors [4]–[6] and reprojection error that is formulated as distance on the image plane, they are robust to image noise, outliers and lighting variations.

With the advent of General Purpose GPU (GPGPU) programming whole-image alignment has become cheap and accessible. This has spawned a new set of methods that use all image pixels and estimate a *dense* and more useful reconstruction of the observed scenes [7]. Due to the increased computational demands, dense methods simplify the inference framework by discarding variable cross-correlations and alternating between tracking and mapping. More pixel information used in the estimation process allows for increased robustness to image degradation due to motion blur, but the reliance on photometric error measured in pixel intensity makes dense methods fragile to sequences violating the brightness constancy assumption.

The success of deep learning inspired a variety of new systems. Learned components have been integrated into the SLAM pipeline to a differing degree, with solutions ranging from classical model based systems delegating sub-components to a neural network (e.g. [8], [9]) to purely end-to-end learned systems that directly predict the camera poses and the geometry of the scene (e.g. [10]–[12]). Priors learned directly from data allow solving difficult tasks such as depth prediction from monocular imagery [13]–[16].

We present *DeepFactors*, a real-time SLAM system that builds and maintains a dense reconstruction but allows for probabilistic inference and combines the advantages of different SLAM paradigms. It also presents a tight integration of learning and model based methods through a learned compact dense code representation that drives significant changes to the core mapping/tracking components of the SLAM pipeline. The main

Manuscript received September 10, 2019; accepted December 23, 2019. Date of publication January 9, 2020; date of current version January 22, 2020. This letter was recommended for publication by Associate Editor V. Indelman and Editor S. Behnke upon evaluation of the reviewers' comments. This work was supported by Dyson Technology Ltd. *(Corresponding author: Jan Czarnowski.)*

The authors are with the Dyson Robotics Laboratory, Imperial College London, SW7 2AZ, London, U.K. (e-mail: j.czarnowski15@imperial.ac.uk; t.laidlow15@imperial.ac.uk; ronald.clark@imperial.ac.uk; ajd@doc.ic.ac.uk).

This article has supplementary downloadable material available at https://ieeexplore.ieee.org, provided by the authors.

Digital Object Identifier 10.1109/LRA.2020.2965415

contributions presented in this letter can be summarised as follows:

- The first real-time probabilistic dense SLAM system,
- A system that integrates learned priors over geometry with classical SLAM formulations in a probabilistic factor-graph formulation.

II. Related Work

In CodeSLAM [17], the authors have introduced the concept of learning a compact optimisable representation and utilising it to solve the dense structure-from-motion problem. The feasibility of this idea has been proved by implementing a windowed 3D reconstruction/visual odometry. The presented system lacked features of full SLAM, was not capable of real-time performance and did not generalise to real handheld camera scenes.

Our work builds on the CodeSLAM idea and explores the impact of the code optimisation on traditional SLAM pipelines. DeepFactors is a complete new SLAM system build from scratch with a different mapping backend, error formulations (the sparse geometric and reprojection errors) and all the various design choices within the SLAM algorithm like keyframing, map maintenance or tracking. It contains features that CodeSLAM was missing – local and global loop closure or relocalisation and optimises the full map in batch instead of a fixed window only.The optimised GPU usage, efficient implementation and SLAM design choices enable real-time performance and the use of a standard factor graph software allows for straightforward probabilistic integration of different sensor modalities which was not possible with the previous formulations of dense SLAM.

An example of other directly comparable work is CNN-SLAM [18]. Although with a substantially different principle of operation, the authors present a real-time full SLAM system that builds a large scale map and supports loop closures by utilising LSD-SLAM [19] and incorporating learned priors. We use CNN-SLAM as a baseline comparison for our method.

There exists a range of systems with learned components which are not fully-featured SLAM systems but instead focus on multi-frame dense reconstruction [8], [20]–[22]. In BA-Net [23] the authors use a technique similar to CodeSLAM where a set of basis depth maps is predicted from the image and optimised in a bundle adjustment problem to find a dense per-pixel reconstruction. The system has been trained end-to-end with the optimisation included which might result in a learned representation better suited for the structure from motion problem. In contrast to our work, BA-Net is not a real-time SLAM system that builds and optimises a consistent multi-keyframe map.

A notable mention is DeepTAM [24], which builds upon DTAM [7] by replacing both the TV-L1 optimisation and camera tracking with a deep convolutional neural network and achieves results outperforming standard model-based methods. In contrast to our work, it follows the same tracking and mapping split used in all dense methods and is not capable of real-time operation. Although the authors took special care to address the generalisation problem, their system still ultimately relies on seeing all possible variations of input data, which is hard in the case of full 6 DoF motion in real world conditions. Our approach relies less on the network generalisation as we perform optimisation that allows to correct for bad network predictions as in our system, neural networks are mainly used for obtaining an image conditioned manifold to optimise over.

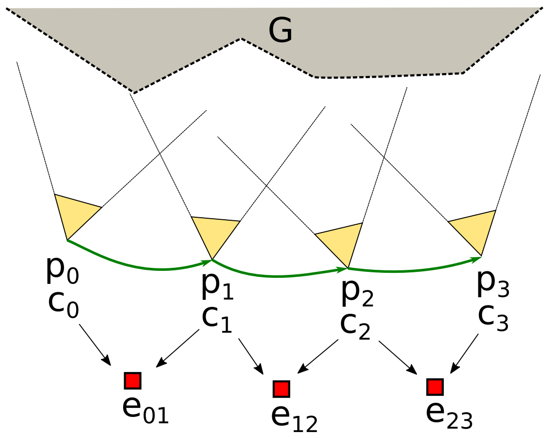

Fig. 2. Dense multi-frame structure from motion problem using an optimisable compact dense code representation. Each camera frame i consists of a 6DoF world pose p_i and an a associated code $\mathbf{c}_i$. We minimise various pairwise consistency losses e_{ij} in order to find the best estimate for scene geometry $G = \{D_0(c_0), \ldots, D_n(c_n)\}$ and camera poses $p_0 \ldots p_N$.

In contrast to the previously mentioned methods, our work strives towards the goal of a unified SLAM framework that incorporates both learned and model-based methods as well as dense and sparse approaches to localisation and mapping and possibly points towards a new generation of SLAM systems. In the remainder of our paper, we explain the building blocks of our system and show evaluation on real world sequences.

III. Code Based Optimization

To reconstruct a dense representation of the scene geometry and estimate the camera motion we formulate a multi-view dense bundle adjustment problem. We parametrise the reconstructed geometry G as a set of depth maps at each camera frame $G = \{\mathrm{D}_0, \mathrm{D}_1, \ldots, \mathrm{D}_n\}$. In a naïve formulation, pixels of each depth are uncorrelated and optimised independently, which makes the problem too ill-posed and costly to solve due to the large number of parameters.

Following the methodology proposed in [17] we optimise depth on a learned compact manifold (code) to mitigate both of these problems (Fig. 2). We express the depth map D_i of a frame i as a function of code $\mathbf{c}_i$ and the associated image $\mathtt{I}_i$. In order to avoid costly relinearisations during optimisation, we require this relation to be linear:

$$\mathtt{D}_i = f(\mathbf{c}_i, \mathtt{I}_i) = \mathtt{D}_i^0 + \mathtt{J}(\mathtt{I}_i)\mathbf{c}_i, \qquad (1)$$

where $\mathtt{D}_i^0 = f(\mathbf{0}, \mathtt{I}_i)$ is the depth map resulting from decoding an all-zero code and $\mathtt{J}(\mathtt{I}_i) = \frac{\partial \mathtt{D}_i}{\partial \mathbf{c}_i}$ is the image-conditioned Jacobian.

When optimising on the code manifold, groups of depth pixels are correlated together which makes the optimisation problem more tractable. Fig. 3 presents the pixels affected by perturbing different elements of the latent code vector.

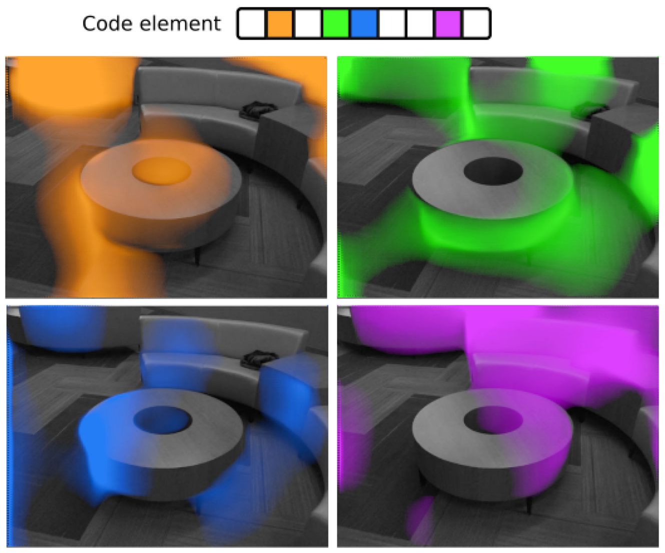

Fig. 3. Visual representation of the code Jacobian $\frac{\partial D_i}{\partial \mathbf{c}_i}$. Perturbing different code elements affects various regions of the depth image.

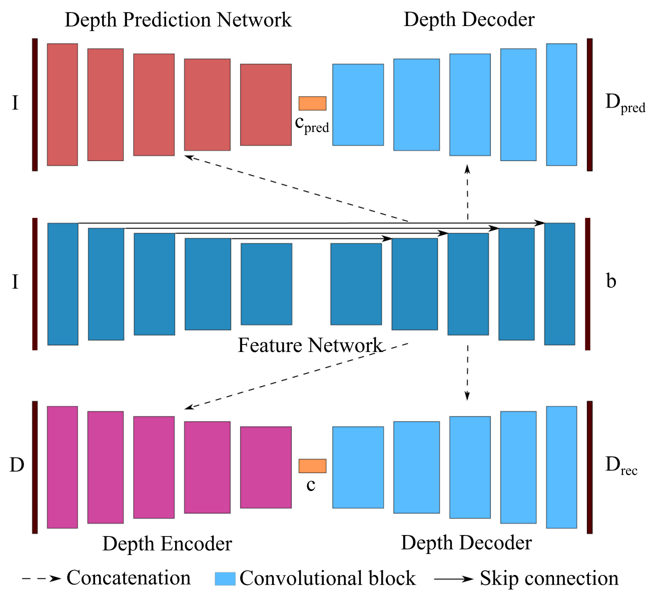

Fig. 4. Network architecture. The bottom path is a U-Net depth auto-encoder without skip connections that learns the optimisable compact code c. Its encoder and decoder are conditioned on image features concatenated from the Feature Network. The top part of the network learns to predict optimal code c_{pred} from the input RGB image that is decoded into a mono depth prediction. The depth decoder is shared between the two networks (light blue).

Using the code representation, we minimise a set of different objective functions that ensure consistency between observations from the overlapping camera frames and find the best estimate of scene geometry and camera motion. These functions are used as pairwise constraints in the factor graph used by our SLAM system. The following sections describe these factors in greater detail.

A. Photometric Factor

A consistency loss typically used in Dense SLAM is the photometric error. It measures the difference directly between source image intensities $\mathtt{I}_i$ and the target image $\mathtt{I}_j$ warped into frame i:

$$e^{ij}_{pho} = \sum_{\mathbf{x}\in\Omega_i} ||\mathtt{I}_i(\mathbf{x}) - \mathtt{I}_j(\omega_{ji}(\mathbf{x}, \mathbf{c}_i, \mathtt{I}_i))||^2, \tag{2}$$

where ω_{ji} warps pixel coordinates $\mathbf{x}$ in frame i to frame j:

$$\omega_{ji}(\mathbf{x}, \mathbf{c}_i, \mathtt{I}_i) = \pi(\mathtt{T}_{ji}(\pi^{-1}(\mathbf{x}, \mathtt{D}_i(\mathbf{x})))), \tag{3}$$

where π and π^{-1} are the projection and reprojection function respectively, $\mathtt{D}_i(\mathbf{x}) = \mathtt{D}(\mathbf{x}, \mathbf{c}_i, \mathtt{I}_i)$ is the depth map decoded from code $\mathbf{c}_i$ and $\mathtt{T}_{ji} \in SE(3)$ is the relative 6DoF transformation from frame i to j.

B. Reprojection Factor

We also use the indirect reprojection error widely used in classical structure from motion. Given a set of matched landmark observations between the images, this method measures the differences between their observed and hypothesised locations:

$$e^{ij}_{rep} = \sum_{(\mathbf{x},\mathbf{y})\in M_{ij}} ||\omega_{ji}(\mathbf{x}, \mathbf{c}_i, \mathtt{I}_i) - \mathbf{y}||^2, \tag{4}$$

where M_{ij} is a set of salient image features matched between frame i and j. To handle mismatched features, we use Cauchy robust cost function that has a constant response to outliers. We use BRISK [6] to detect and describe key points in images.

C. Sparse Geometric Factor

Another form of consistency can be expressed with differences in scene geometry. In our simplified form, we compare depth map D_j with depth map D_i warped into frame j:

$$e^{ij}_{geo} = \sum_{\mathbf{x}\in\Omega_i} ||[\mathtt{T}_{ji}(\pi^{-1}(\mathbf{x}, \mathtt{D}_i(\mathbf{x})))]_z - \mathtt{D}_j(\hat{\mathbf{x}})||^2, \tag{5}$$

where $\hat{\mathbf{x}} = \omega_{ji}(\mathbf{x}, \mathbf{c}_i, \mathtt{I}_i)$ and $[\mathbf{x}]_z$ denotes taking the z component of the vector $\mathbf{x}$. We use Huber norm [25] on the error as a robust cost function. In order to save computation, we evaluate the loss only for a sparse set of uniformly sampled pixels. It is possible to sample a different set of pixels at each iteration to stochastically optimise the loss over the whole image.

IV. Network Architecture

For learning the compact code representation we modify the network from CodeSLAM [17]. The full network architecture is presented in Fig. 4. The middle part represents a U-Net [26] extracting features from the input RGB image. The input is processed with blocks of convolutions with each block reducing the size and applying multiple convolutions on the reduced resolution. The bottom part of the figure depicts the main Variational Auto-Encoder(VAE)[27] that learns the optimisable compact depth representation. The encoder and decoder are conditioned on the above-mentioned features using concatenations.

Similar to CodeSLAM, in order to keep the relation between the reconstructed depth D_{rec} and the code c linear, we do not use any non-linear activations in the depth decoder. To also ensure

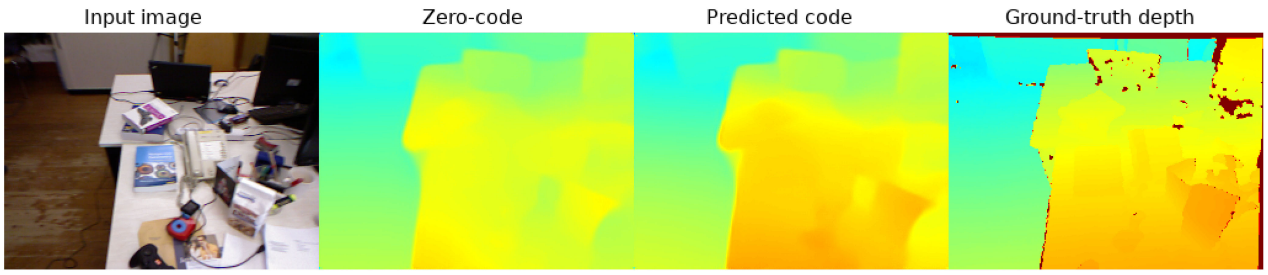

Fig. 5. A comparison of two ways of initialising new depth maps given an input intensity image: zero-code and explicit code prediction. The inclusion of an additional network that predicts an initial code directly from the image allows for better initial depth estimates.

that the input image retains influence on that relation, we add an element-wise multiplication of each two concatenated layers in the conditioning concatenations.

Due to the KL-divergence based latent loss applied to the code of the depth VAE, an all-zero code corresponds to a likely depth map for the input image I. In our experiments with running the system on a real camera, we have found that we can achieve better initial depth predictions by augmenting the network with a separate encoder that explicitly predicts an optimal code from the input image I. For the predicted code to lie in the same space as the learned code c, we apply the same latent loss to it. The added explicit network path is less constrained than the zero code prediction, which allows to achieve better results, as shown in Fig. 5.

The network also predicts an uncertainty parameter b with the Feature Network which is used in a multi-resolution negative log of Laplacian likelihood loss for the reconstructed depth D_{rec}:

$$\sum_{x\in\Omega} \frac{|\mathtt{D}_{rec}(x) - \mathtt{D}(x)|}{b(x)} + \log(b(x)),$$

where Ω is the set of all pixel coordinates of the input depth D. The predicted depth map is supervised with an L1 loss:

$$\sum_{x\in\Omega} |\mathtt{D}_{pred}(x) - \mathtt{D}(x)|. \tag{6}$$

V. System

The system builds and maintains a keyframe map. Incoming new camera images are resized and corrected to match the network focal length and tracked against the nearest keyframe (Section V-A). Once sufficient baseline and other criteria are met, a new keyframe is initialised at the estimated pose with an initial code prediction and added to the graph (Section V-C).

To optimise the map, we maintain a factor graph of the batch MAP problem and optimise it each time new observations are introduced. Each new keyframe is connected to last N keyframes in the map using selected pairwise consistency factors presented in Section III. Real-time performance of solving for the batch solution is achieved by using an incremental mapping algorithm (Section V-B).

To increase performance, the system alternates between tracking and joint mapping. After creating a new keyframe, the graph is being optimised for a set number of iterations or until convergence. During that time the tracking and mapping optimisation

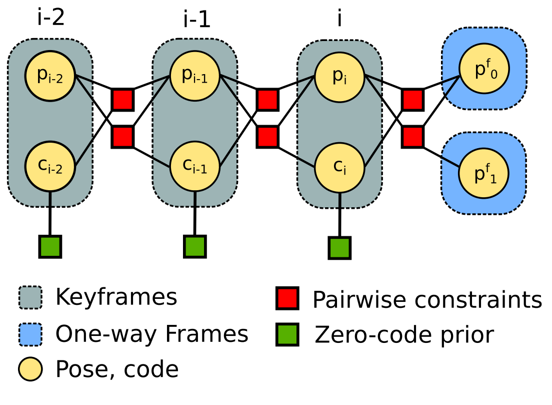

Fig. 6. An example instance of the factor graph used in our system containing three keyframes and two one-way frames attached to the latest keyframe. Multiple different pairwise constraints can be used simultaneously (photometric, geometric or reprojection) but for simplicity, only a single type has been presented in this figure.

steps are interleaved to ensure that the system keeps up with incoming new camera images.

To obtain good quality photometric signal a mixture of low and high baseline image pairs is required. Since we only connect last N keyframes with pairwise constraints, this might not always be the case. To mitigate this while keeping the computational complexity low we introduce "one-way" frames that do not have attached depth and are used to feed information to refine the latest keyframe. After optimising for a set amount of steps, active one-way frames are marginalised and removed from the graph. This allows for inexpensive integration of many views into the optimisation and quality reconstruction of depth.

The details of each component of the system have been described in the following sections.

A. Camera Tracking

Each camera frame is tracked against the closest keyframe using our GPGPU implementation of a standard direct whole-image SE3 Lucas-Kanade [28]–[30]. In case tracking is lost, we perform relocalization by attempting to align a small resolution image against all keyframes.

B. Incremental Keyframe Mapping

We formulate and jointly optimise a batch MAP estimation problem involving all keyframes in the map. Fig. 6 presents a factor graph representation of an example instance of a map built by our system. Each keyframe is represented by a pose p_i and a code c_i variable with variables of neighbouring keyframes involved in pairwise consistency factors (photometric, reprojection or geometric factors). Since the factors were designed to represent a single-way warping between two keyframes, they allow optimisation of the code/depth of only a single keyframe of the pair. Two factors are added to optimise both keyframes. Because during training the code manifold is enforced to be close to a zero-mean Gaussian (variational latent loss), the code has to be kept within the appropriate region during optimisation by using zero-code prior factors that regularise it.

The mapping step performs batch optimisation of all keyframes in the map using standard factor graph software [31]. To make optimisation feasible in large scale scenarios we rely on an incremental mapping algorithm – iSAM2 [32]. It stores a factorisation of the batch Jacobian in the form of a Bayes Tree and incrementally updates it when new variables are added. Only the affected factors are relinearised and re-factorised, which greatly limits the computation required for obtaining the batch solution and allows near constant time updates in an exploration-type movement.

In order to enable marginalisation of one-way frames, we enforce a specific elimination order of the graph variables to ensure that they are leaves in the clique tree built and maintained by iSAM2.

C. Initialization

We use the explicit code prediction network to obtain an initial code for each new keyframe. The depth decoder is then used to transform the codes into a depth map estimate. On start, the system can be trivially initialised in the exact same manner. The network prediction must be good enough for tracking the initial camera movement so that new views can be fed into the system to refine the depth prediction.

In cases when the single image prediction fails a multi-frame initialisation can be used, in which a keyframe is created for each input frame and the initial graph is optimised before starting the system. We found the single frame initialisation to perform well enough for most scenes.

D. Loop Closure

We detect and close local and global loops. Within an active region of last 10 keyframes, we assume the relative estimated poses to be correct and therefore use a pose-based criteria to detect loops and add additional pairwise constraints between keyframes. This tightens the graph and allows for more consistent local reconstructions.

At each new input frame we also test for global loop closure events. Outside of the active window we assume that too much drift has been accumulated and therefore we cannot use our estimates to close loops. Initial loop candidates are detected using a bag of words approach [33] and later further eliminated by attempting to track the current frame against each of them and checking the resulting number of inliers and the estimated pose distance. Because of the photometric error typically having a smaller convergence basin we add only reprojection factors when closing a detected global loop between two keyframes.

TABLE I
COMPARISON OF THE INFLUENCE OF DIFFERENT FACTOR TYPES ON THE ESTIMATED TRAJECTORY AND RECONSTRUCTION ERRORS
(*pho:* PHOTOMETRIC. *geo:* GEOMETRIC, *rep:* REPROJECTION)

Factor Comparison				
Sequence	**Factors Used**	**ATE RMSE↓**	**absrel↓**	**pc110↑**
scene0565_00	pho	0.128	0.108	57.56%
	pho+rep	**0.112**	0.104	59.80%
	pho+geo	0.115	0.103	59.75%
	combined	0.114	**0.102**	**60.13%**
scene0084_00	pho	0.131	0.085	69.14%
	pho+rep	0.074	0.082	71.05%
	pho+geo	0.120	0.081	71.81%
	combined	**0.061**	**0.077**	**73.66%**
scene0606_02	pho	0.089	0.214	37.22%
	pho+rep	0.071	0.201	39.39%
	pho+geo	0.067	0.168	44.45%
	combined	**0.066**	**0.162**	**46.16%**

VI. Experimental Results

A. Training

We have trained our network on a fragment of the ScanNet dataset [34] following the official training/test split. The depth data comes from a real sensor and therefore contains missing values. Since the dataset provides PLY models of the full scene reconstructions together with ground truth poses, we have rendered depth maps from the models and combined them with the raw sensor data to obtain final merged depth maps. We used around 1.4 M images in total.

The network was trained for 13 epochs with learning rate 0.0001, image size 256×192 and code size 32. This is the maximum code size that we can achieve real-time results with in our SLAM system.

B. Ablation Studies

DeepFactors combines three different error metrics/factors – photometric, geometric and reprojection to estimate the camera trajectory and the observed scene geometry. In order to determine how each factor type influences the system performance, we have evaluated various configurations on the quality of the estimated trajectory and reconstruction. The photometric factor is treated as a baseline system and the other two types are included both individually and together. We perform this evaluation on selected shortened scenes from the validation set of the ScanNet dataset and use three error metrics: RMSE of the Absolute Trajectory Error (ATE-RMSE), the absolute relative depth difference (absrel)[13] and the average percentage of pixels in which the estimated depth falls within 10% of the true value (pc110) [18]. The results are presented in Table I.

The inclusion of either of the factors reduces the trajectory error and increases the reconstruction accuracy, with the reprojection factor typically having a stronger influence on the former and the geometric factor on the latter. Explicit feature matching utilised within the reprojection error improves local minima

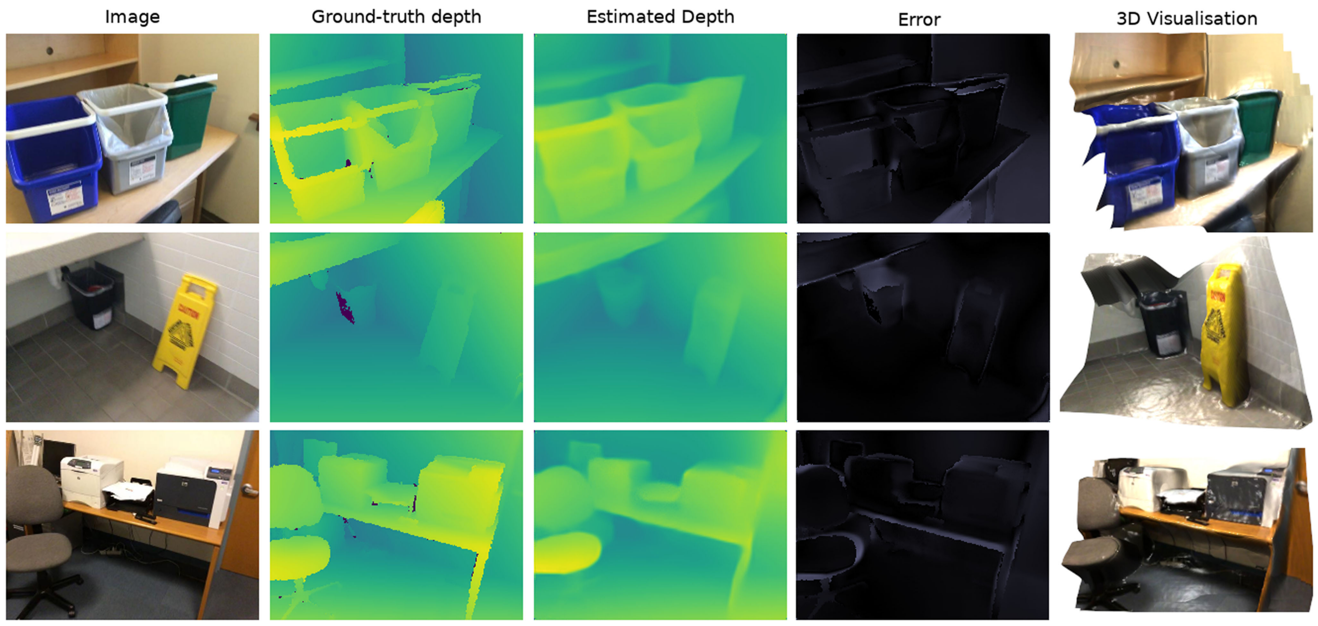

Fig. 7. Example reconstructions on selected scenes from the validation set of the ScanNet dataset. The specular lighting in the 3D visualisation has been exaggerated to better highlight structure. Best viewed on a computer screen.

TABLE II
EVALUATION OF AVERAGE PERCENTAGE OF PIXELS FOR WHICH THE ESTIMATED DEPTH FALLS WITHIN 10% OF THE TRUE VALUE ON THE ICL-NUIM AND TUM DATASETS (TUM/SEQ1: *fr3_ long_ office _household*, TUM/SEQ2: *fr3_ nostructure _texture_ near withloop*, TUM/SEQ3: *fr3 _structure _texture_ far*)

Perc. Correct Depth [%]				
Sequence	Ours	CNN-SLAM [18]	LSD-BS [19]	Laina [37]
icl/office0	**30.17**	19.41	0.60	17.19
icl/office1	20.16	**29.15**	4.76	20.84
icl/living0	**20.44**	12.84	1.44	15.01
icl/living1	**20.86**	13.04	3.03	11.45
tum/seq1	**29.33**	12.48	3.80	12.98
tum/seq2	16.92	**24.08**	3.97	15.41
tum/seq3	**51.85**	27.40	6.45	9.45
Avg.	**27.10**	19.77	3.44	14.62

avoidance and increases convergence rate, which adds robustness to the system. The geometric error introduces a prior about the world that only a single surface is observed and pins separate depth maps together to form a single reconstruction in the textureless areas that lack photometric information. Combining all three factors achieves the best trajectory and reconstruction results.

C. Reconstruction

We evaluate our reconstruction accuracy against CNN-SLAM as it is the only relevant system that evaluates reconstruction using the whole estimated trajectory and without using the ground-truth poses. Since the authors do not provide implementation of their system, we follow their evaluation strategy by taking the results reported in their paper and using the same sequences from the ICL-NUIM [35] and TUM [36] datasets. For all the keyframes produced by each system, we their estimated depth against the ground-truth by calculating the percentage of the pixels for which the depth is within 10% of the true value. The results are presented in Table II.

Since our system is monocular and does not produce up-to-scale trajectories and reconstructions, we use the optimal scale calculated with the TUM benchmark scripts to scale both the trajectory and the depth maps (as done in e.g. [11]).

We outperform all compared methods on most of the sequences and on the average. Our system performs worse on the tum/seq2 trajectory (*fr3_nostructure_texture_near_withloop*) as the camera observes a flat textured wall and due to the small code size (32) used in our system we are typically not able to represent fully flat depth easily.

We also visually present example reconstructions created by our system in Fig. 7.

D. Trajectory Estimation

We evaluate the Absolute Trajectory Error (ATE) of our proposed system and compare it against CNN-SLAM and CodeSLAM. For the sake of completeness, we also include comparison with DeepTAM, despite the fact that it does not achieve interactive (real-time) performance. We have used the same version of CodeSLAM that was used to generate the results in its respective paper. The code for CNN-SLAM is not available and the authors of DeepTAM do not provide a combined tracking and mapping system and we were not able to reproduce the results reported in the letter. For this reason, we include the numbers from the DeepTAM paper and evaluate our system on the same set of trajectories. We omit the room and plant

TABLE III
EVALUATION OF OUR SYSTEM ON THE VALIDATION SEQUENCES OF THE TUM RGB-D BENCHMARK [36].WE COMPARE THE ATE RMSE[m] AGAINST CNN-SLAM AND DeepTAM, RESULTS FOR WHICH HAVE BEEN TAKEN FROM [24]. CNN-SLAM WAS RUN WITHOUT POSE GRAPH OPTIMISATION AND OUR SYSTEM WITHOUT LOOP CLOSURES. BOTH CNN-SLAM AND DeepTAM WERE INITIALISED WITH THE NETWORK FROM CNN-SLAM AND WE USED OUR OWN INITIALISATION. WE'VE OMITTED THE ROOM AND PLANT SEQUENCES AS THEY CONTAIN SIGNIFICANT FRAME DROPS THAT MIGHT SKEW THE RESULTS ***NOT REAL-TIME PERFORMANCE**

Abs. Trajectory Error [m]				
Sequence	Ours	CNN-SLAM	DeepTAM*	CodeSLAM*
fr1/360	0.142	0.500	**0.116**	0.165
fr1/desk	0.119	0.095	**0.078**	0.654
fr1/desk2	0.091	0.115	**0.055**	0.181
fr1/rpy	**0.047**	0.261	0.052	0.078
fr1/xyz	0.064	0.206	**0.054**	0.170

sequences as they contain significant number of dropped frames, which skew the results. In order to limit computation we have disabled the geometric factor for the evaluation.

The results are presented in Table III. We outperform CodeSLAM in all of the sequences and CNN-SLAM in all but one. In most cases we also achieve comparable results to DeepTAM, while still maintaining real-time performance.

VII. PERFORMANCE AND IMPLEMENTATION

For a visual demonstration of the speed of the system please see the associated video, which contains real-time video recordings of our system in action.

Our SLAM system has been implemented in C++ with the dense image warping, optimisation and camera tracking offloaded to GPU with CUDA, while the reprojection and the sparse geometric error factors are computed on the CPU. We run the network, CUDA kernels and visualization on a single NVIDIA GTX 1080 GPU and use image resolution of 256×192.

The network is ran on initialisation of each keyframe in order to obtain an initial code (depth) prediction and the Jacobian $\frac{\delta \mathbf{D}}{\delta \mathbf{c}}$ that is used later in optimisation. This requires around 340 ms, with only 16 ms of it spent on the forward pass of the network and the rest on calculating the Jacobian using *tf.gradients*. This is due to the inefficiency of its backward-mode auto-differentiation based implementation which is optimised for gradients of scalar functions commonly used in machine learning. In our case, obtaining the derivative of each output pixel with respect to the latent representation requires a significant amount of passes through the network. This time can be drastically reduced with engineering effort and we predict it should be possible to reduce the overall time required to run the network to around 30 ms.

The incoming new camera images are tracked against the latest keyframe at around 250 Hz. Once the system initialises a next keyframe, we optimise the whole map representation in batch until convergence. The mapping steps are interleaved with tracking the camera in order to keep up with new images.

The overall performance of the system varies greatly depending on the amount of connectivity between neighbouring keyframes specified by the user, the types of factors enabled, the number of factors relinearised within the iSAM2 algorithm and the occurrence of loop closures. With an explorative-type motion we achieve interactive real-time speeds, where we typically limit our system to only use the photometric and reprojection error to allow it to keep up with the fast camera movement. In a local reconstruction scenario like tabletop AR or room scale reconstruction where there is less exploration we can enable the geometric error to obtain better quality reconstructions. It is possible to further speed up the system performance through engineering effort which is part of planned future work.

The implementation of our system will be released on GitHub and shall be available under the following link: https://github.com/jczarnowski/DeepFactors

VIII. CONCLUSION

We have presented DeepFactors, a real-time probabilistic dense SLAM system built using the the concept of learned compact depth map representation. We have demonstrated that our system achieves greater robustness and precision by combining different paradigms from classical SLAM with priors learned from data in a standard factor-graph probabilistic framework. The use of a standard framework allows it to be easily extended with different sensor modalities, which was not previously possible in the context of purely dense SLAM. An efficient C++ implementation and careful choices in the SLAM design enable real-time performance.

In future, we would like to explore the idea of including the structure-from-motion optimisation within the compact depth code training. This could allow obtaining a code manifold that is specifically trained to be later used in a mapping environment. Moreover, learning the code representation in an unsupervised manner based on the intensity images only could be an interesting experiment. An inclusion of a relative-pose prediction network could also robustify the camera tracking.

We would also like to work on improving the performance of the current system, focusing on a faster method of obtaining the network Jacobian and a better GPU implementation of the geometric factor.

ACKNOWLEDGMENT

The authors would like to thank M. Bloesch for useful insights during the initial stages of the project.

REFERENCES

[1] A. J. Davison, "Real-time simultaneous localisation and mapping with a single camera," in *Proc. Int. Conf. Comput. Vision*, 2003, pp. 1403–1410.

[2] G. Klein and D. W. Murray, "Parallel tracking and mapping for small AR workspaces," in *Proc. Int. Symp. Mixed Augmented Reality*, 2007, pp. 225–234.

[3] 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.

[4] D. G. Lowe, "Distinctive image features from scale-invariant keypoints," *Int. J. Comput. Vision*, vol. 60, no. 2, pp. 91–110, 2004.

[5] E. Rublee, V. Rabaud, K. Konolige, and G. Bradski, "ORB: An efficient alternative to SIFT or SURF," in *Proc. Int. Conf. Comput. Vision,* 2011, pp. 2564–2571.

[6] S. Leutenegger, M. Chli, and R. Siegwart, "BRISK: Binary robust invariance scalable keypoints," in *Proc. Int. Conf. Comput. Vision*, 2011, pp. 2548–2555.

[7] R. A. Newcombe, S. Lovegrove, and A. J. Davison, "DTAM: Dense tracking and mapping in real-time," in *Proc. Int. Conf. Comput. Vision*, 2011, pp. 2320–2327.

[8] C. S. Weerasekera, Y. Latif, R. Garg, and I. Reid, "Dense monocular reconstruction using surface normals," in *Proc. IEEE Int. Conf. Robot. Autom.*, 2017, pp. 2524–2531.

[9] M. Pizzoli, C. Forster, and D. Scaramuzza, "REMODE: Probabilistic, monocular dense reconstruction in real time," in *Proc. IEEE Int. Conf. Robot. Autom.*, 2014, pp. 2609–2616.

[10] R. Clark, S. Wang, H. Wen, A. Markham, and N. Trigoni, "VINet: Visual-inertial odometry as a sequence-to-sequence learning problem," in *Proc. Nat. Conf. Artif. Intell.*, 2017, pp. 3995–4001.

[11] T. Zhou, M. Brown, N. Snavely, and D. G. Lowe, "Unsupervised learning of depth and ego-motion from video," in *Proc. IEEE Conf. Comput. Vision Pattern Recognit.*, 2017, pp. 6612–6619.

[12] Z. Yin and J. Shi, "GeoNet: Unsupervised learning of dense depth, optical flow and camera pose," in *Proc. IEEE Conf. Comput. Vision Pattern Recognit*, 2018, pp. 1983–1992.

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

[14] B. Ummenhofer *et al.*, "DeMoN: Depth and motion network for learning monocular stereo," pp. 5622–5631, 2016, doi: 10.1109/CVPR.2017.596.

[15] F. Liu, C. Shen, and G. Lin, "Deep convolutional neural fields for depth estimation from a single image," in *Proc. IEEE Conf. Comput. Vision Pattern Recognit.*, 2015, pp. 5162–5170.

[16] R. Garg, V. K. B. G, G. Carneiro, and I. Reid, "Unsupervised CNN for single view depth estimation: Geometry to the rescue," in *Proc. Eur. Conf. Comput. Vision*, 2016, pp. 740–756.

[17] M. Bloesch, J. Czarnowski, R. Clark, S. Leutenegger, and A. J. Davison, "CodeSLAM—Learning a compact, optimisable representation for dense visual SLAM," in *Proc. IEEE Conf. Comput. Vision Pattern Recognit.*, 2018, pp. 2560–2568.

[18] 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. Vision Pattern Recognit*, 2017, pp. 6565–6574.

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

[20] T. Laidlow, J. Czarnowski, and S. Leutenegger, "DeepFusion: Real-time dense 3D reconstruction for monocular SLAM using single-view depth and gradient predictions," in *Proc. IEEE Int. Conf. Robot. Autom.*, 2019, pp. 4068–4074.

[21] R. Clark, M. Bloesch, J. Czarnowski, S. Leutenegger, and A. J. Davison, "LS-Net: Learning to solve nonlinear least squares for monocular stereo," in *Comput. Vision ECCV 2018. Lecture Notes Comput. Sci.*, V. Ferrari, M. Hebert, C. Sminchisescu, Y. Weiss, Eds., Springer, Cham, vol. 11212, 2018.

[22] C. Liu, J. Gu, K. Kim, S. G. Narasimhan, and J. Kautz, "Neural RGB->D sensing: Depth and uncertainty from a video camera," *CVPR*, 2019, *arXiv:1901.02571*.

[23] C. Tang and P. Tan, "BA-Net: Dense bundle adjustment network," *ICLR*, 2019, *arXiv:1806.04807*.

[24] H. Zhou, B. Ummenhofer, and T. Brox, "DeepTAM: Deep tracking and mapping," in *Proc. Eur. Conf. Comput. Vision*, 2018, pp. 851–868.

[25] P. J. Huber, "Robust estimation of a location parameter," *Annals Math. Statist.*, vol. 35, no. 1, pp. 73–101, 1964.

[26] O. Ronneberger, P. Fischer, and T. Brox, "U-Net: Convolutional networks for biomedical image segmentation," in *Proc. Int. Conf. Med. Image Comput Comput. Assist. Intervention*, 2015, pp. 234–241.

[27] D. P. Kingma and M. Welling, "Auto-encoding variational bayes," in *Proc. Int. Conf. Learn. Represent.*, 2014. [Online]. Available: https://iclr.cc/archive/2013/conference-proceedings.html

[28] B. D. Lucas and T. Kanade, "An iterative image registration technique with an application to stereo vision," in *Proc. Int. Joint Conf. Artif. Intell.*, 1981, pp. 674–679.

[29] R. A. Newcombe, "Dense visual SLAM," Ph.D. dissertation, Imperial College London, London, U.K., 2012.

[30] S. J. Lovegrove, "Parametric dense visual SLAM," Ph.D. dissertation, Imperial College London, London, U.K., 2011.

[31] F. Dellaert, "Factor graphs and GTSAM," Georgia Institute of Technology, Tech. Rep. GT-RIM-CP&R-2012-002, 2012.

[32] M. Kaess, H. Johannsson, R. Roberts, V. Ila, J. Leonard, and F. Dellaert, "iSAM2: Incremental smoothing and mapping using the Bayes tree," *Int. J. Robot. Res*, vol. 31, no. 2, pp. 216–235, 2012.

[33] D. Gálvez-López and J. D. Tardós, "Bags of binary words for fast place recognition in image sequences," *IEEE Trans. Robot.*, vol. 28, no. 5, pp. 1188–1197, Oct. 2012.

[34] A. Dai, A. X. Chang, M. Savva, M. Halber, T. Funkhouser, and M. Nießner, "ScanNet: Richly-annotated 3D reconstructions of indoor scene," in *Proc. IEEE Conf. Comput. Vision Pattern Recognit.*, 2017, pp. 2432–2443.

[35] A. Handa, T. Whelan, J. B. McDonald, and A. J. Davison, "A benchmark for RGB-D visual odometry, 3D reconstruction and SLAM," in *Proc. IEEE Int. Conf. Robot. Autom.*, 2014. [Online]. Available: http://www.doc.ic.ac.uk/ ahanda/VaFRIC/iclnuim.html

[36] J. Sturm, N. Engelhard, F. Endres, W. Burgard, and D. Cremers, "A benchmark for the evaluation of RGB-D SLAM systems," in *Proc. IEEE/RSJ Conf. Intell. Robots Syst.*, 2012, pp. 573–580.

[37] I. Laina, C. Rupprecht, V. Belagiannis, F. Tombari, and N. Navab, "Deeper depth prediction with fully convolutional residual networks," 2016, doi: 10.1109/3DV.2016.32.