Real-Time Height Map Fusion using Differentiable Rendering

Jacek Zienkiewicz, Andrew Davison and Stefan Leutenegger
Imperial College London, UK

***Abstract*— We present a robust real-time method which performs dense reconstruction of high quality height maps from monocular video. By representing the height map as a triangular mesh, and using efficient differentiable rendering approach, our method enables rigorous incremental probabilistic fusion of standard locally estimated depth and colour into an immediately usable dense model. We present results for the application of free space and obstacle mapping by a low-cost robot, showing that detailed maps suitable for autonomous navigation can be obtained using only a single forward-looking camera.**

I. Introduction

Advancing commodity processing power, particularly from GPUs, has enabled various recent demonstrations of real-time dense reconstruction from monocular video. Usually the approach has been to concentrate on high quality multi-view depth map reconstruction, and then fusion into a generic 3D representation such as a TSDF, surfel cloud or mesh. Some of the systems presented in this vein have been impressive, but we note that there are few examples of moving beyond *showing* real-time dense reconstruction towards *using* it, in-the-loop, in applications. The live reconstructions are often in a form which needs substantial further processing before they could be useful for any in-the-loop use such as path planning or scene labelling.

In this paper, we argue that real-time dense reconstruction can be made much more useful by adopting a more restrictive, task-oriented model and performing incremental fusion directing in this form. In particular, our own interest is low-cost robotics. There are strong economic, power and design reasons which would make it advantageous for a platform such as a small cleaning robot to have a full navigation solution based on monocular vision. While there are now practical robots such as the Dyson 360 Eye which perform localisation using sparse visual SLAM, free-space finding and obstacle avoidance on this and other similar robots is achieved using additional specialised sensors. The default view going forward is that depth cameras or stereo systems will be needed for high quality local perception. To challenge this view, in this paper we show that a generative approach to height mapping can be used in real-time to build dense surface reconstructions suitable for online robot path planning. The only input we use is monocular video from a front-facing RGB camera looking down at an angle towards the floor.

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

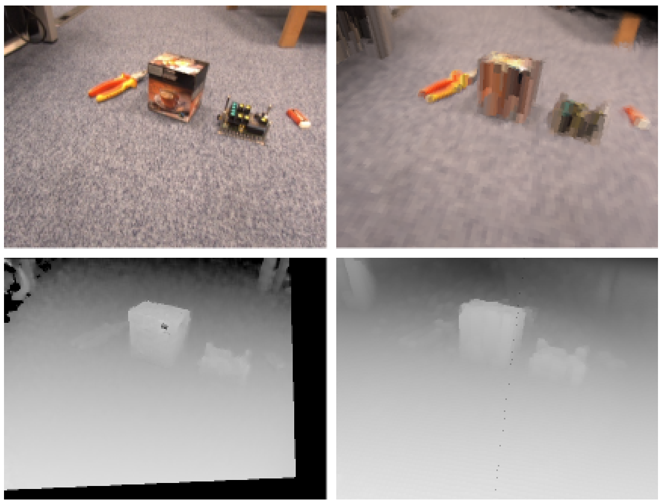

Fig. 1: Left: an example input monocular RGB image and motion stereo depth map which are inputs to our fusion algorithm. Right: output RGB and depth image from differentiable renderer.

Our core representation for reconstruction is a coloured height map defined as a triangular mesh on a regular grid. At each new video frame, we first estimate a depth map using a simple multi-view stereo algorithm. Next, given a current estimate of the surface parameters, we perform a predictive colour and depth rendering and compared it with observed image and depth map. The errors between the observations and predictions together with the gradients calculated using differentiable rendering are used for iterative optimisation of the parameters of each observed surface cell. After the optimisation terminates, we fuse the current measurement by updating the per-triangle quadratic cost functions, our representation of uncertainty and priors that summarise the information contained in the previous images.

Because our reconstruction is already in the height map representation which is directly relevant to ground robotics, simple thresholding of the mean height is sufficient to generate usable quantities such as the drivable free-space area or a classification of walls, furniture and small obstacles based on colour and height. To prove that this is a genuinely practical way to use real-time dense mapping, we present results from a mobile robot platform which moves through and maps a variety of typical indoor environments. We also present a comparison against a generic 3D mapping technique that relies on high quality depth maps.

II. Related work

Real-time dense tracking and mapping, using parallel computation from GPUs, is an increasingly practical technique since the publication of KinectFusion [18], but this and the majority of other methods, *e.g.* [13], [26], have required depth camera input. Monocular RGB methods like DTAM [19] have generally not been nearly as robust. Depth reconstruction from RGB requires dense multi-view stereo matching, which is computationally demanding and has accuracy which varies widely depending on lighting levels and the amount of texture in a scene. For this reason, the most usable monocular SLAM systems are still those which performance sparse or semi-dense reconstruction only, *e.g.* [5], [17]. Some demonstrations have been made of the use of dense monocular vision for in-the-loop robotic control [7], but so far in simplified or artificially highly textured settings.

To address the challenges of efficiency and robustness, it is often advantages to take a more application-directed approach to the representation and fusion, and use *e.g.* height maps. There exists significant prior work on height map estimation both in the field of robotics [10], [14] as well as computer vision [2]. Height maps were successfully applied to terrain mapping and mobile robot navigation using laser scanners [21], [27], as well as stereo and depth cameras [8], [9], [28]. However, typically height map fusion approaches are data-driven and rely on heuristics to solve the data association problem, *e.g.* in [6] only the highest measurement per height cell is fused into the height map, whereas remaining values are discarded. Furthermore, in order to make the fusion computationally tractable the dependencies between the individual elements of the map are often ignored.

Very relevant to our approach is the early work of Cernuschi-Frias *et al.* [3], as well as the approach by Chesseman *et al.* [4] and the results on Bayesian shape recovery from NASA [12], [23], [24]. The key characteristic of these methods is that the estimation is not performed in images domain (*e.g.* in stereo matching search is performed along epipolar lines) but directly in parameters space of the model. The surface reconstruction is formulated as an inverse problem: given a set of images infer the most probably surface that could have generated them. Surface is typically modelled using parametrised shapes or on a discrete uniform grid, and Bayes theorem is used to derive a formal solution to this problem. Hung *et al.* [11] extended the framework and derived a sequential Bayesian estimator for a parametrised surface model. Similar to our algorithm, information extracted from previous images is summarised in a quadratic form. However, compared to the described approaches which infer the surface properties directly from RGB images, we estimate the surface more robustly by rendering and fusing the depth maps.

Our method is also inspired by OpenDR [15], where Loper and Black demonstrated how the standard computer graphics rendering pipeline can be use to obtain generative models suitable for solving various computer vision problems.

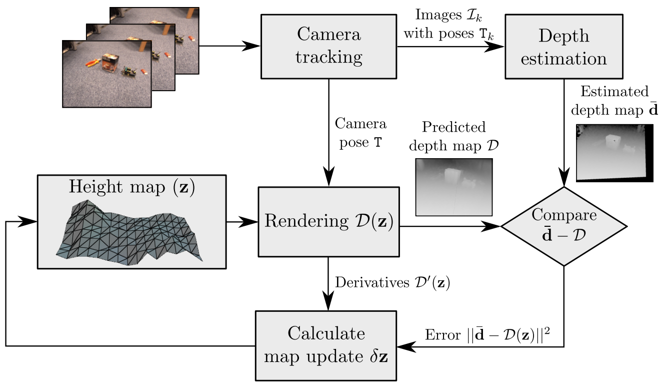
Camera tracking
Images $\mathcal{I}_k$ with poses $\mathtt{T}_k$
Depth estimation
Estimated depth map $\bar{\mathbf{d}}$
Camera pose $\mathtt{T}$
Predicted depth map $\mathcal{D}$
Height map $(\mathbf{z})$
Rendering $\mathcal{D}(\mathbf{z})$
Compare $\bar{\mathbf{d}} - \mathcal{D}$
Derivatives $\mathcal{D}'(\mathbf{z})$
Calculate map update $\delta\mathbf{z}$
Error $||\bar{\mathbf{d}} - \mathcal{D}(\mathbf{z})||^2$

Fig. 2: An overview of our system. Given a sequence of images from a single moving camera we continuously estimate the camera motion and use it for a depth estimation. Using differentiable rendering, the calculated depth maps are fused into a height map represented by a triangular mesh.

III. Method

Given a sequence of images from a camera moving over a scene, our goal is to recursively estimate the camera trajectory and a surface model defined as a fixed-topology triangular mesh above a regular grid. To simplify the problem, we apply the common separation of camera tracking and mapping: in a first step, only the camera motion is estimated, which is subsequently treated as a fixed quantity. We track camera frame-to-frame motion precisely using dense image alignment [19] in a similar fashion to [30]. Subsequently we perform straightforward depth map estimation using a small subset (5-10) of most recent frames and a multi-view stereo algorithm based on a plane sweep and cost volume filtering [22]. The depth maps are feed into our incremental mapping module in a loosely-coupled approach. Rather than spending the effort on improving and denoising the depth maps, our system allocates computational resources to the fusion process. Note that our fusion approach is more general and can be used in conjunction with any camera tracking (*e.g.* ORB-SLAM [17]) and depth estimation method (*e.g.* stereo or Kinect camera).

Incremental reconstruction is formulated as a recursive nonlinear optimisation problem, where as each new frame arrives we compare it with a generative rendering of our current surface estimate and make an appropriate Bayesian update. In this respect, we formulate nonlinear residuals as the difference between the (inverse) depths as measured in the current frame and the (inverse) depth predictions generated by the rendered model; at the same time, we compare predicted colours and rendered colours. In order to obtain a recursive formulation that allows us to keep *all* past measurements, we linearise these error terms and keep them as priors that are jointly minimised with the residuals of the current frame. Fig. 2 shows an overview of our method.

An important element of our method which enables highly efficient operation is a differentiable renderer implemented within the standard OpenGL computer graphics pipeline.

Given a current surface model and a camera pose it can render a predicted image and depth for each pixel together with the derivatives of these quantities with respect to the model parameters at almost no extra computational cost.

The key strengths of our method are as follows:

- Probabilistic interpretation and a generative model. We perform incremental Bayesian fusion using a per triangle information filter. The approach is optimal up to linearisation errors and discards no information, while the computational complexity is bounded.
- Scalability both in terms of image resolution and scene representation. Using current GPUs, rendering can be done extremely efficiently, and calculating derivatives comes at almost negligible cost.
- Robustness and efficiency comes from working directly in the state space of interest for applications such as mobile robotics.

In the following we formalise our fusion method.

A. *The Generative Model*

Our approach is inspired by the probabilistic model of the image formation and rendering process illustrated in Fig. 3. A surface is parametrised by its geometry $\mathtt{G}$ and its appearance $\mathtt{A}$. Given a camera with an associated pose $\mathtt{T}$ in the scene, we can render a predicted image $\mathcal{I}$ and an inverse depth map $\mathcal{D}$. In our method we do not model lighting and surface properties (such as normals) explicitly, but assume ambient light and Lambertian surfaces.

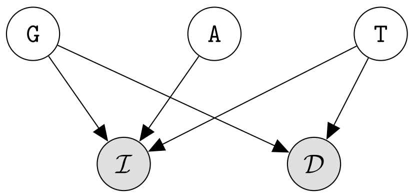

Fig. 3: A graphical model of the image formation and rendering process used to derive our fusion approach.

The joint distribution that models the image formation process in Fig. 3 is given by:

$$P(\mathcal{I},\mathcal{D},\mathtt{G},\mathtt{A},\mathtt{T}) = P(\mathcal{I}|\mathtt{G},\mathtt{A},\mathtt{T})P(\mathcal{D}|\mathtt{G},\mathtt{T})P(\mathtt{G})P(\mathtt{A})P(\mathtt{T}) \ . \tag{1}$$

The relation between image observations and surface estimation can be expressed using Bayes rule:

$$P(\mathtt{G},\mathtt{A},\mathtt{T}|\mathcal{I},\mathcal{D}) \propto P(\mathcal{I},\mathcal{D}|\mathtt{G},\mathtt{A},\mathtt{T})P(\mathtt{G})P(\mathtt{A})P(\mathtt{T}) \ , \tag{2}$$

which allows us to derive a maximum *a posteriori* (MAP) estimate of the camera pose and surface:

$$\arg\max_{\mathtt{G},\mathtt{A},\mathtt{T}} P(\mathcal{I},\mathcal{D}|\mathtt{G},\mathtt{A},\mathtt{T})P(\mathtt{G})P(\mathtt{A})P(\mathtt{T}) \ . \tag{3}$$

The term $P(\mathcal{I},\mathcal{D}|\mathtt{G},\mathtt{A},\mathtt{T})$ is a likelihood function which we will be able to evaluate and differentiate using our renderer. The terms $P(\mathtt{G})$, $P(\mathtt{A})$, $P(\mathtt{T})$ represent prior knowledge that we might have about the geometry, appearance and trajectory.

To simplify the problem, we treat the camera poses as given by a tracking module. Furthermore, since in our loosely-coupled fusion, we use the colour images to generate a depth map that determines the height field, we ignore the dependency of the colour image on the height values. Note that this is a conservative assumption letting us treat the colours and height fields independently:

$$\arg\max_{\mathtt{G}} P(\mathcal{D}|\mathtt{G})P(\mathtt{G}) \ , \tag{4a}$$

$$\arg\max_{\mathtt{A}} P(\mathcal{I}|\mathtt{A})P(\mathtt{A}) \ . \tag{4b}$$

In essence, we alternate between height (Eq. 4a) and colour fusion (Eq. 4b), first estimating the geometry using the observed inverse depth map and subsequently we fuse the colour image into the height fields while keeping the geometry fixed. In the following we focus on the depth map fusion, but the derivation can be extended to the colour estimation in a straightforward manner. Currently, the colour information is mainly used for meaningful display, however we plan to use it to perform camera tracking with respect to the estimated model in order to reduce the tracking drift.

B. *Reconstruction as a Nonlinear Least Squares Problem*

The geometry of a height field is fully parametrised by the vector of height $\mathbf{z} \in \mathbb{R}^n$, which is the quantity we want to estimate. We represent the priors $P(\mathtt{G})$ using a multivariate Gaussian probability distribution $\mathcal{N}^{-1}(\boldsymbol{\eta}, \Lambda)$ in a canonical form parametrised by the information vector $\boldsymbol{\eta}$ and matrix Λ. By taking the negative logarithm of Eq. 4a we obtain the following minimisation problem:

$$\arg\min_{\mathbf{z}} F(\mathbf{z}) \ , \tag{5}$$

where:

$$F(\mathbf{z}) = F_d(\mathbf{z}) + F_p(\mathbf{z}) \ . \tag{6}$$

The cost function thus consists of two terms, the data term $F_d(\mathbf{z})$ and the prior term $F_p(\mathbf{z})$:

$$F_d(\mathbf{z}) = \|\bar{\mathbf{d}} - \mathcal{D}(\mathbf{z})\|^2_{\Sigma_{\bar{\mathbf{d}}}} \ , \tag{7a}$$

$$F_p(\mathbf{z}) = \frac{1}{2}\mathbf{z}^\top \Lambda \mathbf{z} - \boldsymbol{\eta}^\top \mathbf{z} + c \ , \tag{7b}$$

where $\bar{\mathbf{d}} \in \mathbb{R}^m$ is a column vector that represents the observed inverse depth map with associated measurement uncertainties modelled by (diagonal) covariance matrix $\Sigma_{\bar{\mathbf{d}}}$, and $\mathcal{D}(\mathbf{z})$ is a nonlinear (rendering) operator, $\mathcal{D} : R^n \to R^m$ that predicts a vector of depths using the current estimate of $\mathbf{z}$. The prior term (Eq. 7b) is a standard quadratic form associated with the Gaussian probability distribution $\mathcal{N}^{-1}(\boldsymbol{\eta}, \Lambda)$.

The cost function term $F_d(\mathbf{z})$ related to the measurements can be expressed as a sum of individual per-pixel inverse depth error terms:

$$e_i(\mathbf{z}) = \bar{d}_i - d_i(\mathbf{z}), \tag{8}$$

between a measured $\bar{d}_i$ and a predicted inverse depth d_i:

$$F_d(\mathbf{z}) = \sum_{i=1}^{m} \frac{1}{\sigma^2_{d_i}} (\bar{d}_i - d_i(\mathbf{z}))^2 = \sum_{i=1}^{m} \frac{1}{\sigma^2_{d_i}} e_i(\mathbf{z})^2 \ . \tag{9}$$

The minimisation problem in Eq. 5 is a nonlinear least squares problem, that is typically solved using iterative approaches. Commonly used method for solving such problems is the Gauss-Newton algorithm, that starts with an initial estimate $\mathbf{z}_0$ and solves a series of linear approximations of the original nonlinear problem. At each iteration, it is required to form a normal equation and solve it *e.g.* by means of Cholesky factorisation. However, due to the size of our problem using direct methods that form an approximation for the Hessian explicitly and rely on matrix factorisation is prohibitively expensive. Instead, we rely on the conjugate gradient algorithm which is an indirect, matrix-free approach that only requires access to the gradient of the objective function. Conjugate gradient descent has been successfully applied to many computer vision problems before, *e.g.* in large scale bundle adjustment [1].

The gradient of the objective function $F(\mathbf{z})$ as defined in Eq. 5 is given by:

$$\boldsymbol{\nabla} F(\mathbf{z}) = \boldsymbol{\nabla} F_d(\mathbf{z}) + \boldsymbol{\nabla} F_p(\mathbf{z}) \ . \tag{10}$$

The gradient of the term associated with the surface priors is straightforward to compute:

$$\boldsymbol{\nabla} F_p(\mathbf{z}) = \Lambda\mathbf{z} - \boldsymbol{\eta} \ , \tag{11}$$

whereas the gradient of the data term is given by:

$$\boldsymbol{\nabla} F_d(\mathbf{z}) = 2\mathcal{D}'(\mathbf{z})^\top \Sigma_{\bar{\mathbf{d}}}^{-1}(\bar{\mathbf{d}} - \mathcal{D}(\mathbf{z})) \ , \tag{12}$$

where $\mathcal{D}'(\mathbf{z})$ is the derivative of the rendering operator $\mathcal{D}$ and is called the Jacobian matrix $\mathtt{J}_d(\mathbf{z})$. Although the number of measurements (m) and size of the state space (n) are high, the Jacobian $\mathtt{J}_d(\mathbf{z}) \in \mathbb{R}^{m\times n}$ linking them is sparse. Since we assume that each depth value depends only on 3 vertices, the $\mathtt{J}_d$ has only 3 non-zero entries per row.

Instead of explicitly forming the Jacobian matrix and performing matrix vector multiplication, we can calculate the gradient vector by summing the contributions from the individual, per-pixel error terms. The gradient of $F_d(\mathbf{z})$ is therefore given by:

$$\boldsymbol{\nabla} F_d(\mathbf{z}) = 2\sum_{i=1}^{m} \frac{1}{\sigma_{d_i}^2} \mathtt{E}_i(\mathbf{z}) e_i(\mathbf{z}) \ , \tag{13}$$

where $\mathtt{E}_i(\mathbf{z}) = \boldsymbol{\nabla} e_i(\mathbf{z}^j)$ is the derivative (Jacobian) of the per-pixel error term. Since each error term depends on three values in $\mathbf{z}$, we only need to calculate derivative of the rendered depth with respect to vertices of the associated triangle, and can assemble the gradient of the function F_d from this individual per depth Jacobians $\mathtt{E}_i$. In the following subsection we will describe our strategy to obtain the $\mathtt{E}_i$.

C. Differentiable Rendering

In our rendering we explicitly model the ray/triangle intersection and perform analytical differentiation of this operation. Furthermore we assume that each pixel "observes" only a single triangle. There exist alternative and more sophisticated approaches, *e.g.* Smelyanskiy *et al.* [23] carefully models the rendering process taking into consideration surface normals and contributions from multiple triangles into pixel colour, whereas Loper and Black [15] proposed an approximate way for calculation of the derivative.

Let $\mathbf{r}(t)$ be a ray, parametrised by its starting point $\mathbf{p} \in \mathbb{R}^3$ and direction vector $\mathbf{d} \in \mathbb{R}^3$, $\mathbf{r}(t) = \mathbf{p} + t\mathbf{d}$, with $t \geq 0$. For each pixel in the image we can calculate a ray using camera intrinsics and the center of camera frame of reference as the origin. Let $\triangle$ be a triangle in $\mathbb{R}^3$ parametrised by 3 vertices, $\mathbf{v}_0$, $\mathbf{v}_1$, $\mathbf{v}_2$. To identify the triangle a ray is intersecting, we rely on the standard OpenGL rendering pipeline.

Ray/triangle intersection can be easily found using *e.g.* the classical algorithm of Möller and Trumbore [16], which, when the ray intersects the triangle, yields a vector $(t, u, v)^\top$, where t is the distance to the plane in which the triangle lies and u, v are the barycentric coordinates of the ray intersection point with respect to the triangle $\triangle$. The t, u, and v are the essential elements required to render a depth and colour for a particular pixel: t is directly related to the depth, where as the barycentric coordinates are used to interpolate the colour $\mathbf{c}$ based on the RGB colour triangle vertices $(\mathbf{c}_0, \mathbf{c}_1, \mathbf{c}_2)$ in the following way:

$$\mathbf{c} = (1 - u - v)\mathbf{c}_0 + u\mathbf{c}_1 + v\mathbf{c}_2 \ . \tag{14}$$

Fig. 4 depicts this ray-triangle intersection problem.

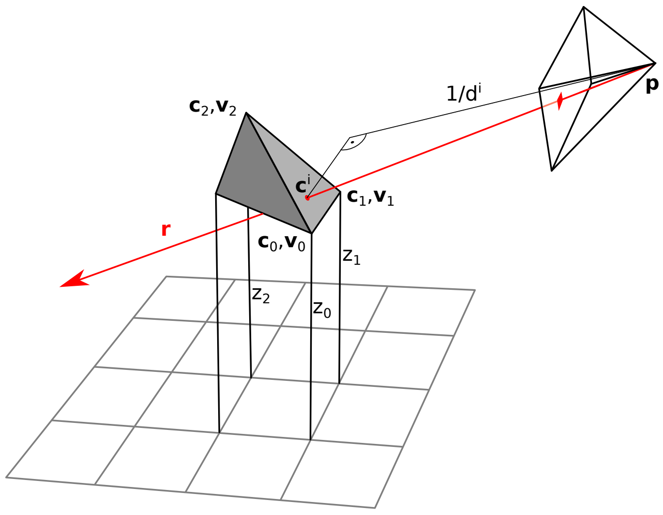

Fig. 4: The essential geometry of ray/triangle intersection.

The rendered inverse depth d_i of pixel i depends only on the geometry of the triangle that a ray is intersecting (and camera pose that is assumed fixed). As we model the surface using a height map, each vertex has only one degree of freedom, its height z. Assuming that the ray intersects the triangle j specified by heights z_0, z_1, z_2, at distance $1/d_i$, we can express the derivative as follows:

$$\mathtt{E}_i = \frac{\partial d_i}{\partial \mathbf{z}_j} = \left[\frac{\partial d_i}{\partial z_0} \ \frac{\partial d_i}{\partial z_1} \ \frac{\partial d_i}{\partial z_2}\right] \ . \tag{15}$$

The individual partial derivatives can be derived easily using chain and product rule, and we omit it due to space limitation.

D. Nonlinear Conjugate Gradient

The objective function (Eq. 5) is optimised using the Fletcher and Reeves variant of the conjugate gradient method [20]. The Alg. 1 outlines the overall structure of this algorithm.

Algorithm 1 Nonlinear conjugate gradient, Fletcher and Reeves version.

```
 1: Given $\mathbf{z}_0$
 2: Evaluate $F_0 = F(\mathbf{z}_0)$ and $\mathbf{g}_0 = \nabla F(\mathbf{z}_0)^\top$    ▷ Render
 3: Set $\mathbf{p}_0 := -\mathbf{g}_0$ and $k := 0$
 4: while $\mathbf{g}_k \neq 0$ do
 5:     Do line search to determine step size $\alpha_k$    ▷ Render
 6:     $\mathbf{z}_{k+1} = \mathbf{z}_k + \alpha_k \mathbf{p}_k$
 7:     Calculate gradient $\mathbf{g}_{k+1} = \nabla F(\mathbf{z}_{k+1})^\top$    ▷ Render
 8:     $\beta_{k+1} := \frac{\mathbf{g}_{k+1}^\top \mathbf{g}_{k+1}}{\mathbf{g}_k^\top \mathbf{g}_k}$
 9:     $\mathbf{p}_{k+1} := -\mathbf{g}_{k+1} + \beta_{k+1}\mathbf{p}_k$
10:     $k := k + 1$
11: end while
```

The ▷ *Render* in lines 2, 5 and 7 in the Alg. 1 highlights the execution of the differentiable rendering. At each iteration of the conjugate gradient method it is required to perform a line search that determines the step size α_k in the descent direction. This can lead to several evaluations of the cost function. Since when evaluating the cost function within the same parallelised rendering code we can instantaneously access its gradient, we do not search for the optimal step size, but accept any α_k that leads to a decrease in the cost. The method typically required several iterations to converge and the key to the efficiency of our method is the very fast differentiable rendering.

E. Height Field Fusion Through Linearisation

Our fusion method is both simple and principled. We accumulate all previous observations in the form of the prior term $F_p(\mathbf{z})$ as a quadratic cost function that serves as constraints on the vertices during the optimisation described in Sec. III-B.

After optimisation converges we linearise the objective function $F_d(\mathbf{z})$ associated with the data term at the estimated solution $\hat{\mathbf{z}}$:

$$F_d(\mathbf{z}) \approx F_d^l(\mathbf{z}) = \|\bar{\mathbf{d}} - \left(\hat{\mathbf{d}} + \hat{\mathtt{J}}_d \cdot (\hat{\mathbf{z}} - \mathbf{z})\right)\|^2 \ , \quad (16)$$

where $\hat{\mathbf{d}} = \mathbf{d}(\hat{\mathbf{z}})|_{\mathbf{z}=\hat{\mathbf{z}}}$ and $\hat{\mathtt{J}}_d = \mathtt{J}_d(\mathbf{z})|_{\mathbf{z}=\hat{\mathbf{z}}}$. Note that we omit here the measurement covariance matrix $\Sigma_{\bar{\mathbf{d}}}$ for the brevity of the derivation. The quadratic approximation of the objective function is therefore given by:

$$F_d^l(\mathbf{z}) = \|\overbrace{\left(\bar{\mathbf{d}} - \hat{\mathbf{d}} - \hat{\mathtt{J}}_d\hat{\mathbf{z}}\right)}^{\mathbf{r}} + \hat{\mathtt{J}}_d\mathbf{z})\|^2 = \|\mathbf{r} + \hat{\mathtt{J}}_d\mathbf{z}\|^2 \quad (17a)$$

$$= \mathbf{r}^\top\mathbf{r} + 2\mathbf{r}^\top\hat{\mathtt{J}}_d\mathbf{z} + \mathbf{z}^\top\hat{\mathtt{J}}_d^\top\hat{\mathtt{J}}_d\mathbf{z} \quad (17b)$$

To fuse the depth measurement vector $\bar{\mathbf{d}}$ into the height field, we simply augment the prior term in Eq. 7b using the quadratic model $F_d^l(\mathbf{z})$ derived above:

$$\Lambda^+ = \Lambda + \hat{\mathtt{J}}_d^\top\hat{\mathtt{J}}_d \ , \quad (18a)$$

$$\boldsymbol{\eta}^+ = \boldsymbol{\eta} + 2\mathbf{r}^\top\hat{\mathtt{J}}_d \ , \quad (18b)$$

$$c^+ = c + \mathbf{r}^\top\mathbf{r} \ . \quad (18c)$$

This operation is equivalent to the measurement update step in the Extended Information Filter [25]. Note that we do not have to keep the value of linearisation point, nor the previous depth measurements.

In practices, we do not perform the steps in Eq. 18 explicitly (which would require forming the matrix $\hat{\mathtt{J}}_d^\top\hat{\mathtt{J}}_d$), but again revert to the per-pixel inverse depth error terms as defined in Eq. 7a and accumulate the *per-pixel* quadratic costs, on a *per-triangle* basis. This means that for each triangle j we keep a quadratic function of the form:

$$f_j(\mathbf{z}) = \mathbf{z}^\top\Lambda_j\mathbf{z} + \boldsymbol{\eta}_j^\top\mathbf{z} + c_j \ . \quad (19)$$

Recall the individual per-pixel error term as in Eq. 8. As already explained, after the optimisation has converged, we approximate this error term linearly around the current estimate $\hat{\mathbf{z}}$ as:

$$e_i \approx e_i^l = \bar{e}_i + \mathbf{E}_i\delta\mathbf{z} = \bar{e}_i - \mathbf{E}_i\hat{\mathbf{z}} + \mathbf{E}_i\mathbf{z} \ . \quad (20)$$

Fusion of a depth measurement $\bar{d}_i$ into the height map thus consists of a simple addition of the linearised error (Eq. 20) to the corresponding triangle's cost function (Eq. 19):

$$f_j^+ = f_j + \frac{(e_i^l)^2}{\sigma_{d_i}^2}. \quad (21)$$

Multiplying this out and rearranging provides us with the updated coefficients (c^+, $\boldsymbol{\xi}^+$ and Λ^+) of the per-triangle quadratic cost:

$$\Lambda_j^+ = \Lambda_j + \frac{\mathbf{E}_i^\top\mathbf{E}_i}{\sigma_{d_i}^2} \ , \quad \boldsymbol{\xi}_j^+ = \boldsymbol{\xi}_j + \frac{2}{\sigma_{d_i}^2}(\bar{e}_i - \mathbf{E}_i\hat{\mathbf{z}})\mathbf{E}_i \ ,$$
$$c_j^+ = c_j + \frac{(\bar{e}_i - \mathbf{E}_i\hat{\mathbf{z}})^2}{\sigma_{d_i}^2}. \quad (22)$$

The prior cost (Eq. 7b) concerning the height map can be assembled from the individual per-triangle error terms (Eq. 19), and therefore the overall cost function (Eq. 5) we have to minimise amounts to:

$$F(\mathbf{z}) = \sum_j f_j(\mathbf{z}) + \sum_i \frac{1}{\sigma_{d_i}^2}(e_i(\mathbf{z}))^2 \quad (23)$$

Note that, consequently, the number of linear cost terms is bounded by the number of triangles in the height field, whereas the number of nonlinear (inverse) depth error terms is bounded by the number of pixels in the camera. This is of course an important property for real-time operation.

IV. Implementation

The core of the algorithm, including derivative computation is implemented using a standard OpenGL rendering pipeline, and does not rely on vendor-specific frameworks like Nvidia CUDA. Only the operations required by the conjugate gradient, *e.g.* dot product are implemented in CUDA.

Run time	GTX 980	GT 650M
Tracking [ms]	2.18	14.5
Depth Estimation [ms]	5.40	24.7
Fusion [ms]	3.41	26.9
Total run time [ms]	10.99	66.9
Total frame rate [fps]	**91.0**	**15.1**

TABLE I: Timings for the algorithms on two different GPUs.

Thanks to the regular, grid structure of the height map, rendering can be performed very efficiently and robustly: depending on height field and image resolution, we can achieve rendering rates of about 6000 frames per seconds on a high-level GPU (GTX980). For a typical scene, there are usually about 50000-100000 triangles visible in the camera frustum.

Table I shows runtime measurements for the whole algorithm and its individual components on two different GPUs, Nvidia GTX980 and Nvidia GT650M. The latter is a mid-level mobile GPU, comparable in performance with the latest Nvidia embedded board Jetson TX1[1]. We achieve real-time performance on both platforms. Note that the effective frame rate can be even higher, as in our system the *Depth Estimation* and *Fusion* do not have to be executed for each video frame — these are performed only when there is sufficient baseline due to camera motion.

V. Experiments and Results

We experimentally evaluated our height map fusion using both synthetic data and mobile robot moving in indoor environments. In the robot experiments, we have used a single Point Grey BlackFly3 Camera, with approximately 80° field of view, capturing RGB images at 640×480 resolution and 30Hz frame-rate. Our robot platform is a Pioneer 3 DX with an adjustable rigid camera mount. The camera was mounted at a height of about 30 cm above the ground, pointing downwards and looking ahead about 1 metre. The extrinsics of the camera on the robot were determined using calibration from [29]. In most of the experiments we used the cells of the size of 10 mm.

Fig. 5 shows a few examples of the colour images that are inputs to our system, and the depth maps we initially derive from these using fast cost volume filtering. The overall quality of the depth maps is relatively low: this is due to the very fast but unsophisticated stereo algorithm that we apply to obtain the depth, as well as drift in the motion tracking. Whereas most monocular dense algorithms allocate a lot of resources and use sophisticated methods to obtain high quality depth maps, our method focuses on fusion into a model and is capable of dealing with partially corrupted and missing data.

A. Synthetic data

We first evaluate our height fusion algorithm using synthetic *moon* data (Fig. 6) and compare it with a simple height map fusion algorithm that treats all height cells in the map as independent (as for example in [6]). Both methods are capable of accurately reconstructing the surface, however, as our approach models the connectivity between vertices of the height map and can incorporate smoothness priors, it better handles situations that arise when the camera is looking at the scene from a very oblique angle. There, for the points that are far from the camera, the resolution of the height map is often higher than the density of the depth measurements; this leads to reconstruction artefacts when the height vertices are assumed to be independent.

B. Comparison against a generic 3D reconstruction with a depth camera

We qualitatively compare our monocular height map estimation to more generic 3D reconstruction method which uses a depth camera. The system of Whelan *et al.* [26] takes a high quality depth-maps obtained by a Kinect camera using a hand-guided camera trajectory and is capable of

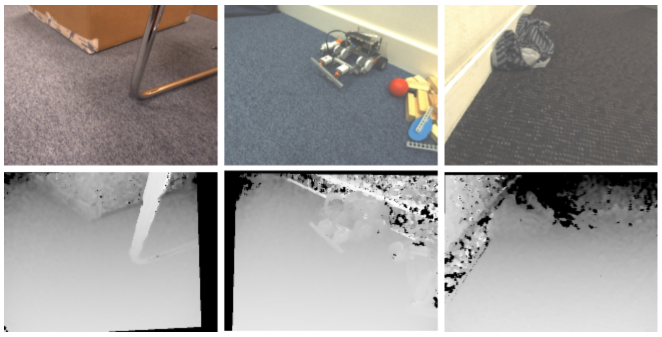
Fig. 5: Examples of input RGB and depth images that are fused into a height map.

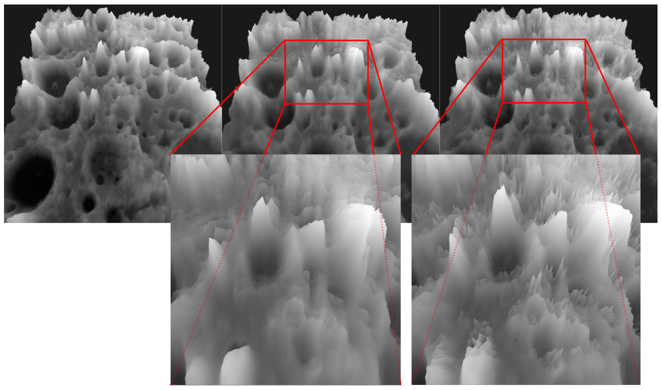
Fig. 6: Evaluation of the fusion algorithm on the synthetic data: (left) ground truth, (middle) reconstruction using proposed method, (right) reconstruction using Fankhauser *et al.* method [6]. In general, both methods are able to accurately reconstruct the surface from just a few depth maps. However, our method better deals with situations where the resolution of the height map is higher compared to density of measurements (*e.g.* due to oblique viewing angle).

[1]http://www.nvidia.com/object/jetson-tx1-module.html. Jetson TX1 has a credit-card footprint and power consumption of about 15W.

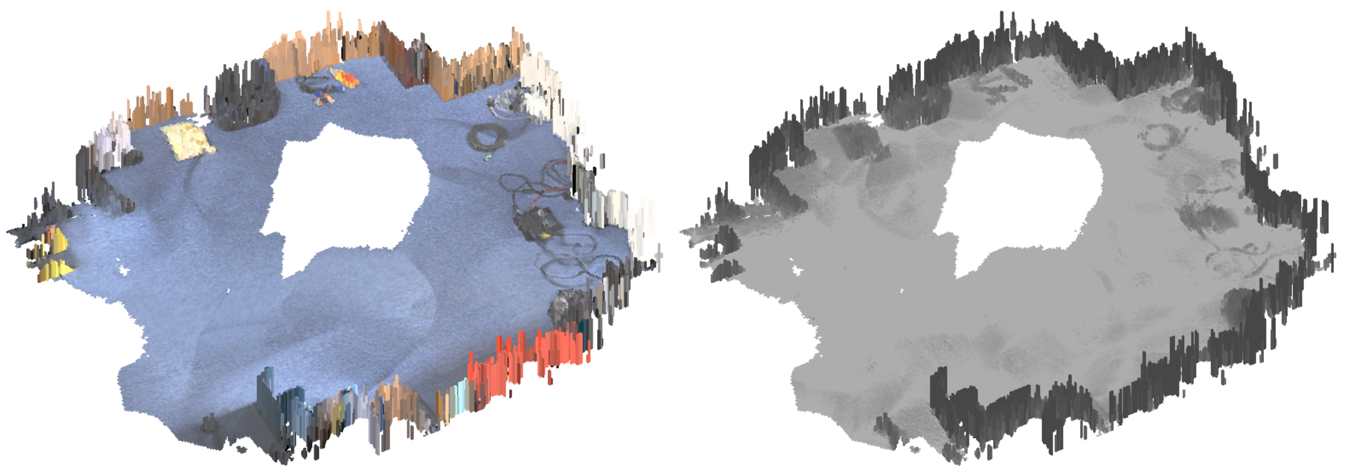

Fig. 7: A visualisation of a height map reconstructed with our method.

creating globally consistent 3D models. Our method, like any incremental, open-loop system is a subject to drift and is not designed to directly compete the ElasticFusion in terms of a global accuracy. However, as shown in Fig. 8 our approach is capable of creating maps of the ground with similar local accuracy. The visualisation in Fig. 7 demonstrates that we were able to reconstruct the essential geometry of the environment as well as the very small objects on the floor, like cables or pliers.

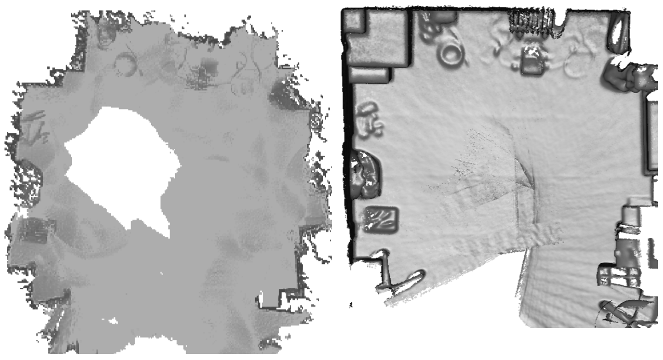

Fig. 8: A comparison of the reconstruction from our height fusion (left) with high quality reconstruction obtained using ElasticFusion (right). ElasticFusion [26] uses depth maps from a Kinect-like depth sensor and is capable of creating globally consistent models, whereas our method is open-loop and based on monocular depth estimation. This comparison demonstrates that our system was capable of estimating the essential geometry of the room, and still reconstructs details on the floor.

C. Free space detection

A desirable property of a height map is that it can be directly used for robot navigation and obstacles avoidance. One can determine the drivable area by simply thresholding the height values. Fig. 9 illustrates results of applying this approach to our reconstruction. There, for each pixel in an image, we check the height of associated grid cell and label it as a free space based on a fixed, 1 cm threshold. The created free space mask is subsequently overlaid onto the observed image. Despite the fact that a height map cannot correctly model overhangs, our approach exhibits desirable behaviour even in these scenarios. The method in its current implementation is robust, especially for the task of free space detection. We attribute this to the smoothing behaviour of the height map representation that we use in our method. More sophisticated approaches could evaluate the gradient of the height field to determine roughness of the terrain and the traversability.

VI. Conclusions

We have demonstrated a promising route to truly usable real-time monocular dense reconstruction which is based on a y chosen height map model, differentiable rendering and rigorous incremental probabilistic fusion.

In our particular application area of low-cost robotics this method offers great promise for simple monocular sensing which can efficiently and robustly capture comprehensive information about free space and obstacles, with the ability to accurately map the small obstacles which are invisible to many other sensors and a potential hazard to small robots.

There is great scope for the expansion of this methodology in both this and other application areas of real-time reconstruction. Our probabilistic framework allows us to model and express many aspects of the image formation process, such as camera intrinsics, radial distortions, rolling shutter, blur, camera gain — and improve accuracy by jointly estimating these quantities.

We are particularly interested to investigate the promising scalability of our method — to the very high resolution fusion that high resolution video would enable (could make height maps with the millimetre precision enabling tiny

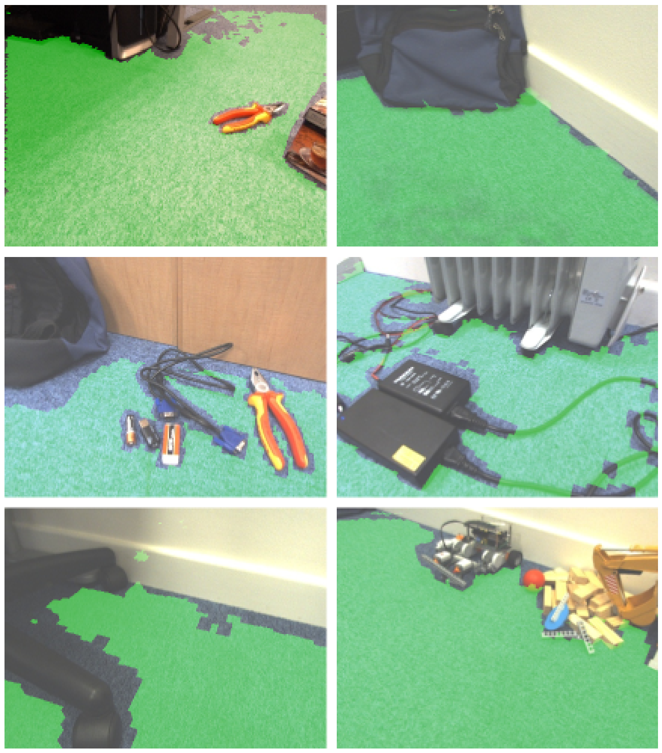

Fig. 9: Free space detected (green overlay) by simple thresholding of the reconstructed elevation map.

objects like pins to be mapped?); but also to the very efficient implementation which should be possible at lower resolution, perhaps for extreme resource-limited platforms. Research on multi-resolution methods might unify these two ends of the scale.

References

[1] S. Agarwal, N. Snavely, S. M. Seitz, and R. Szeliski. Bundle Adjustment in the Large. In *Proceedings of the European Conference on Computer Vision (ECCV)*, 2010. 4

[2] R. Cabezas, O. Freifeld, G. Rosman, and J. W. Fisher III. Aerial Reconstructions via Probabilistic Data Fusion. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2014. 2

[3] B. Cernuschi-Frias, D. B. Cooper, Y.-P. Hung, and P. N. Belhumeur. Toward a model-based Bayesian theory for estimating and recognizing parameterized 3-D objects using two or more images taken from different positions. *IEEE Transactions on Pattern Analysis and Machine Intelligence (PAMI)*, 11(10):1028–1052, 1989. 2

[4] P. Cheeseman, B. Kanefsky, R. Kraft, J. Stutz, and R. Hanson. Super-Resolved Surface Reconstruction from Multiple Images. In *Maximum Entropy and Bayesian Methods*, volume 62, pages 293–308. 1996. 2

[5] J. Engel, T. Schoeps, and D. Cremers. LSD-SLAM: Large-scale direct monocular SLAM. In *Proceedings of the European Conference on Computer Vision (ECCV)*, 2014. 2

[6] P. Fankhauser, M. Bloesch, C. Gehring, M. Hutter, and R. Siegwart. Robot-Centric Elevation Mapping with Uncertainty Estimates. In *Proceedings of the International Conference on Climbing and Walking Robots (CLAWAR)*, 2014. 2, 6

[7] C. Forster, M. Fässler, F. Fontana, M. Werlberger, and D. Scaramuzza. Continuous on-board monocular-vision-based elevation mapping applied to autonomous landing of micro aerial vehicles. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, 2014. 2

[8] D. Gallup, J.-M. Frahm, M. Pollefeys, and E. Zuerich. A Heightmap Model for Efficient 3D Reconstruction from Street-Level Video. In *Proceedings of the International Symposium on 3D Data Processing, Visualization and Transmission (3DPVT)*, 2010. 2

[9] C. Häne, C. Zach, J. Lim, A. Ranganathan, and M. Pollefeys. Stereo depth map fusion for robot navigation. In *Proceedings of the IEEE/RSJ Conference on Intelligent Robots and Systems (IROS)*, 2011. 2

[10] M. Herbert, C. Caillas, E. Krotkov, I. Kweon, and T. Kanade. Terrain mapping for a roving planetary explorer. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, 1989. 2

[11] Y.-P. Hung, D. B. Cooper, and B. Cernuschi-Frias. Asymptotic Bayesian surface estimation using an image sequence. *IJCV*, 6(2):105–132, June 1991. 2

[12] A. Jalobeanu, F. O. Kuehnel, and J. C. Stutz. Modeling Images of Natural 3D Surfaces: Overview and Potential Applications. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition Workshop (CVPRW)*, 2004. 2

[13] M. Keller, D. Lefloch, M. Lambers, S. Izadi, T. Weyrich, and A. Kolb. Real-time 3D Reconstruction in Dynamic Scenes using Point-based Fusion. In *Proc. of Joint 3DIM/3DPVT Conference (3DV)*, 2013. 2

[14] I.-S. Kweon and T. Kanade. High-resolution terrain map from multiple sensor data. *IEEE Transactions on Pattern Analysis and Machine Intelligence (PAMI)*, 14(2):278–292, 1992. 2

[15] M. Loper and M. J. Black. OpenDR: An Approximate Differentiable Renderer. In *Proceedings of the European Conference on Computer Vision (ECCV)*, 2014. 2, 4

[16] T. Möller and B. Trumbore. Fast , Minimum Storage Ray / Triangle Intersection. *Journal of Graphics Tools*, 2(1):21–28, 1997. 4

[17] R. Mur-Artal and J. D. Tardós. ORB-SLAM: Tracking and Mapping Recognizable Features. In *Workshop on Multi View Geometry in Robotics (MVIGRO) - RSS 2014*, 2014. 2

[18] R. A. Newcombe, S. Izadi, O. Hilliges, D. Molyneaux, D. Kim, A. J. Davison, P. Kohli, J. Shotton, S. Hodges, and A. Fitzgibbon. KinectFusion: Real-Time Dense Surface Mapping and Tracking. In *Proceedings of the International Symposium on Mixed and Augmented Reality (ISMAR)*, 2011. 2

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

[20] J. Nocedal and S. Wright. *Numerical Optimization*. Springer, second edition, 2006. 5

[21] P. Pfaff, R. Triebel, and W. Burgard. An Efcient Extension to Elevation Maps for Outdoor Terrain Mapping and Loop Closing. *International Journal of Robotics Research (IJRR)*, 26(2):217–230, 2007. 2

[22] C. Rhemann, A. Hosni, M. Bleyer, C. Rother, and M. Gelautz. Fast cost-volume filtering for visual correspondence and beyond. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2011. 2

[23] V. Smelyanskiy, P. Cheeseman, D. A. Maluf, and R. D. Morris. Bayesian super-resolved surface reconstruction from images. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2000. 2, 4

[24] V. Smelyanskiy, R. Morris, F. Kuehnel, D. A. Maluf, and P. Cheeseman. Dramatic Improvements to Feature Based Stereo. In *Proceedings of the European Conference on Computer Vision (ECCV)*, 2002. 2

[25] S. Thrun, Y. Liu, D. Koller, A. Y. Ng, Z. Ghahramani, and H. Durrant-Whyte. Simultaneous Localization and Mapping with Sparse Extended Information Filters. *International Journal of Robotics Research (IJRR)*, 23(7-8):693–716, 2004. 5

[26] T. Whelan, S. Leutenegger, R. F. Salas-Moreno, B. Glocker, and A. J. Davison. ElasticFusion: Dense SLAM without a pose graph. In *Proceedings of Robotics: Science and Systems (RSS)*, 2015. 2, 6, 7

[27] C. Ye and J. Borenstein. A method for mobile robot navigation on rough terrain. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, 2004. 2

[28] E. Zheng, E. Dunn, R. Raguram, and J.-M. Frahm. Efficient and Scalable Depthmap Fusion. In *Proceedings of the British Machine Vision Conference (BMVC)*, 2012. 2

[29] J. Zienkiewicz and A. J. Davison. Extrinsics Autocalibration for Dense Planar Visual Odometry. *Journal of Field Robotics (JFR)*, 32(5):803–825, 2015. 6

[30] J. Zienkiewicz, R. Lukierski, and A. J. Davison. Dense, Auto-Calibrating Visual Odometry from a Downward-Looking Camera. In *Proceedings of the British Machine Vision Conference (BMVC)*, 2013. 2