∇SLAM: Dense SLAM meets Automatic Differentiation

Krishna Murthy J.*[1,2,3], Ganesh Iyer[5], and Liam Paull†[1,2,3,4]

[1]Université de Montréal, [2]Mila, [3]Robotics and Embodied AI Lab (REAL), [4]Candian CIFAR AI Chair, [5]Carnegie Mellon University

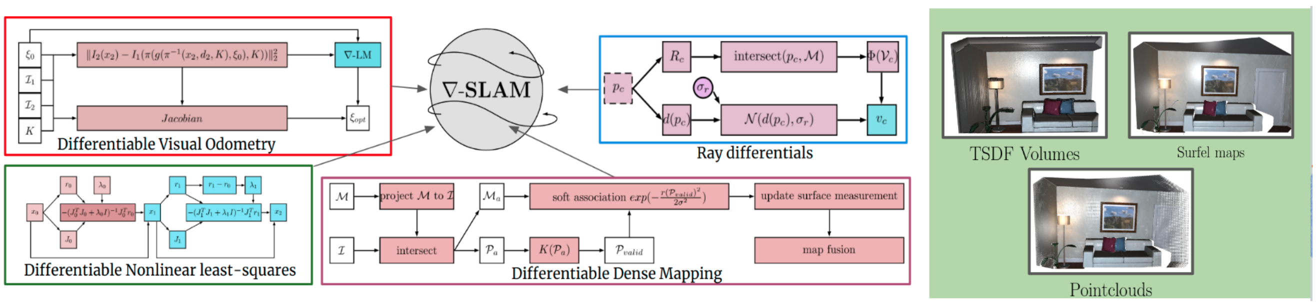

Figure 1. **∇SLAM (gradSLAM)** is a *fully differentiable* dense simultaneous localization and mapping (SLAM) system. The central idea of ∇SLAM is to construct a computational graph representing every operation in a dense SLAM system. By proposing differentiable alternatives to several (usually non-differentiable) components of SLAM (such as optimization, raycasting, etc.), we create a pathway for gradient-flow from 3D map elements to sensor observations (here *pixels*), without impacting performance. We implement differentiable variants of three classical SLAM systems that operate on voxels, surfels, and pointclouds respectively. We are excited about the prospects that ∇SLAM holds for spatially-grounded gradient-based learning (*c.f*. spatial intelligence [9]).

Abstract

The question of "representation" is central in the context of dense simultaneous localization and mapping (SLAM). Newer learning-based approaches have the potential to leverage data or task performance to directly inform the choice of representation. However, learning representations for SLAM has been an open question, because traditional SLAM systems are not end-to-end differentiable.

In this work, we present ***∇SLAM*** ***(gradSLAM)****, a differentiable computational graph take on SLAM. Leveraging the automatic differentiation capabilities of computational graphs, ∇SLAM enables the design of SLAM systems that allow for gradient-based learning across each of their components, or the system as a whole. This is achieved by creating differentiable alternatives for each non-differentiable component in a typical dense SLAM system. Specifically, we demonstrate how to design differentiable trust-region optimizers, surface measurement and fusion schemes, as well as differentiate over rays, without sacrificing performance. This amalgamation of dense SLAM with computational graphs enables us to backprop all the way from 3D maps to 2D pixels, opening up new possibilities in gradient-based learning for SLAM.*

A short video explaining the paper and showcasing the results can be found at https://youtu.be/2ygtSJTmo08.

TL;DR*: We leverage the power of automatic differentiation frameworks to make dense SLAM differentiable.*

1. Introduction

Simultaneous localization and mapping (SLAM) has—for decades—been a central problem in robot perception and state estimation. A large portion of the SLAM literature has focused either directly or indirectly on the question of map representation. This fundamental choice dramatically impacts the choice of processing blocks in the SLAM pipeline, as well as all other downstream tasks that depend on the outpus of the SLAM system. Of late, gradient-based learning approaches have transformed the outlook of several domains (Eg. image recognition [26], language modeling [43], speech recognition [19]). However, such techniques have had limited success in the context of SLAM, primarily since many of the elements in the standard SLAM pipeline

*Correspondence to `krrish94 [at] gmail [dot] com`.
†No ∇ students were harmed in the making of this work.

are not differentiable. A fully differentiable SLAM system would enable task-driven representation learning since the error signals indicating task performance could be back-propagated all the way through the SLAM system, to the raw sensor observations.

This is particularly true for *dense 3D maps* generated from RGB-D cameras, where there has been a lack of consensus on the right representation (pointclouds, meshes, surfels, etc.). Several methods have demonstrated a capability for producing dense 3D maps from sequences of RGB or RGB-D frames [23, 31, 45]. However, none of these methods are able to solve the *inverse mapping* problem, i.e., answer the question: "*How much does a specific pixel-measurement contribute to the resulting 3D map*"? Formally, we desire an the expression that relates a pixel in an image (or in general, a sensor measurement *s*) to a 3D map $\mathcal{M}$ of the environment. We propose to solve this through the development of a differentiable mapping function $\mathcal{M} = \mathcal{G}_{SLAM}(s)$. Then the gradient of that mapping $\nabla_s \mathcal{M}$ can intuitively tell us that *perturbing the sensor measurement* s *by an infinitesimal* δs *causes the map* $\mathcal{M}$ *to change by* $\nabla_s \mathcal{G}_{SLAM}(s) \delta s$.

Central to our goal of realizing a fully differentiable SLAM system are *computational graphs*, which underlie most gradient-based learning techniques. We make the observations that, if an entire SLAM system can be decomposed into elementary operations, all of which are differentiable, we could compose these elementary operations[1] to preserve differentiability. However, modern *dense* SLAM systems are quite sophisticated, with several non-differentiable subsystems (optimizers, raycasting, surface mapping), that make such a construct challenging.

We propose ∇SLAM (*grad*SLAM), a differentiable computational graph view of SLAM. We show how *all* non-differentiable functions in SLAM can be realised as smooth mappings. First, we propose a differentiable trust region optimizer for nonlinear least squares systems. Building on it, we present differentiable strategies of mapping, raycasting, and global measurement fusion.

The ∇SLAM framework is very general, and can be extended to any existing SLAM system and make it differentiable[2]. In Sec. 4, we provide three examples of SLAM systems that can be realized as differentiable computation graphs: implicit-surface mapping (Kinectfusion [31]), surfel-based mapping (PointFusion [23]), and iterative closest point (ICP) mapping (ICP-SLAM). We show that the differentiable approaches maintain similar performance to their non-differentiable counterparts, with the added advantage that they allow gradients to flow through them.

To foster further research on differentiable SLAM systems and their applications to spatially-grounded learning, we will make ∇SLAM available as an open-source framework[3]. For more videos, results, etc. please visit http://montrealrobotics.ca/gradSLAM/.[4]

2. Related Work

Several works in recent years have applied recent machine learning advances to SLAM or have reformulated a subset of *components* of the full SLAM system in a differentiable manner.

2.1. Learning-based SLAM approaches

There is a large body of work in deep learning-based SLAM systems. For example, CodeSLAM [2] and SceneCode [47] attempt to represent scenes using compact *codes* that represent. 2.5D depth map. DeepTAM [48] trains a tracking network and a mapping network, which learn to reconstruct a voxel representation from a pair of images. CNN-SLAM [41] extends LSD-SLAM [11], a popular monocular SLAM system, to use single-image depth predictions from a convnet. Another recent trend has been to try to formulate the SLAM problem over higher level features such as objects, which may be detected with learned detectors [46] [30] [34]. DeBrandandere et al. [3] perform lane detection by backpropagating least squares residuals into a frontend module. Recent work has also formulated the passive [24] and active localization problems [4, 13] in an end-to-end differentiable manner. While all of these approaches try to leverage differentiability in submodules of SLAM systems (eg. odometry, optimization, etc.), there is no single framework that models an entire SLAM pipeline as a differentiable graph.

2.2. Differentiable visual odometry

The beginnings of differentiable visual odometry can be traced back to the seminal Lucas-Kanade iterative matching algorithm [29]. Kerl *et al*. [25][5] apply the Lucas-Kanade algorithm to perform real-time dense visual odometry. Their system is differentiable, and has been extensively used for self-supervised depth and motion estimation [12, 28, 49]. Coupled with the success of Spatial Transformer Netowrks (STNs) [22], several libraries (gvnn [17], kornia [10]) have since implemented these techniques as differentiable *layers*, for use in neural networks.

However, extending differentiability beyond the two-view case (*frame-frame alignment*) is not straightforward. Global consistency necessitates fusing measurements from live frames into a global model (*model-frame alignment*), which is not trivially differentiable.

[1] Again, using differentiable composition operators.

[2] In some cases, analytical or numerical approximations may be required.

[3] An earlier version of this manuscript is under review. We intend to open-source our framework by the end of January, 2020.

[4] The URL is case-sensitive.

[5] The formulation first appeared in Steinbrüker *et al*. [38].

2.3. Differentiable optimization

Some approaches have recently proposed to learn the optimization of nonlinear least squares objective functions. This is motivated by the fact that similar cost functions have similar loss landscapes, and learning methods can help converge faster, or potentially to better minima.

In BA-Net [40], the authors learn to predict the damping coefficient of the Levenberg-Marquardt optimizer, while in LS-Net [5], the authors entirely replace the Levenberg-Marquard optimizer by an LSTM netowrk [20] that predicts update steps. In GN-Net [44], a differentiable version of the Gauss-Newton loss is used to show better robustness to weather conditions. RegNet [16] employs a learning-based optimization approach based on photometric error for image-to-image pose registration. However, all the aforementioned approaches require the training of additional neural nets and this requirement imposes severe limitations on the generalizability. OptNet [1] introduces differentiable optimization layers for quadratic programs, that do not involve learnable parameters.

Concurrently, Grefenstette *et al*. [14] propose to unroll optimizers as computational graphs, which allows for computation of arbitrarily higher order gradients. Our proposed differentiable Levenberg-Marquardt optimizer is similar in spirit, with the addition of gating functions to result in better gradient flows.

In summary, to the best of our knowledge, there is no *single* approach that models the entire SLAM pipeline as a differentiable model, and it is this motivation that underlies ∇SLAM.

3. ∇SLAM

In this section we will overview our proposed method for ∇SLAM and also detail the individual differentiable subcomponents.

3.1. Preliminaries: Computational graphs

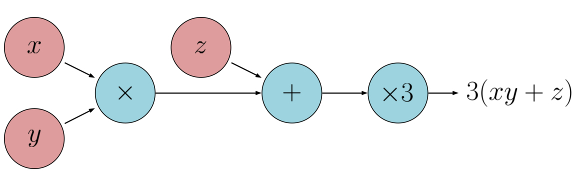

Figure 2. An example computational graph. Nodes in red represent variables. Nodes in blue represent operations on variables. Edges represent data flow. This graph computes the function $3(xy + z)$.

In gradient-based learning architectures, all functions and approximators are conventionally represented as *computational graphs*. Formally, a computation graph is a directed acyclic graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, where each node $v \in \mathcal{V}$ holds an operand or an operator, and each (directed) edge $e \in \mathcal{E}$ indicates the control flow in the graph. Further, each

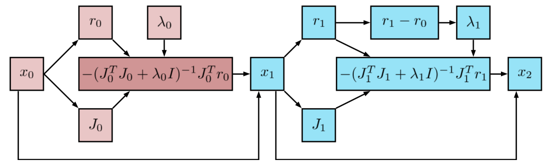

Figure 3. Computational graph for ∇*LM*

node in the graph also specifies computation rules for the gradient of the outputs of the node with respect to the inputs to the node. Computational graphs can be nested and composed in about any manner, whilst preserving differentiability. An example computation graph for the function $3(xy + z)$ is shown in Fig. 2.

In a standard SLAM pipeline there are several subsystems/components that are not differentiable (i.e., for a few forward computations in the graph, gradients are unspecifiable). For example, in the context of dense 3D SLAM [31] [23], nonlinear least squares modules, raycasting routines, and discretizations are non-diffrentiable. Further, for several operations such as index selection / sampling, gradients exist, but are zero *almost everywhere*, which result in extremely sparse gradient flows.

3.2. Method Overview

The objective of ∇SLAM is to make every computation in SLAM exactly realised as a composition of differentiable functions[6]. Broadly, the sequence of operations in dense SLAM systems can be termed as *odometry estimation* (frame-to-frame alignment), *map building* (model-to-frame alignment/local optimization), and *global optimization*. An overview of the approach is shown in 1.

First, we provide a description of the precise issues that render nearly all of the aforementioned modules non-differentiable, and propose differntiable counterparts for each module. Finally, we show that the proposed differentiable variants allow the realization of several classic dense mapping algorithms (*KinectFusion* [31], *PointFusion* [23], ICP-SLAM) in the ∇SLAM framework.[7]

3.3. ∇LM: A differentiable nonlinear least squares solver

Most state-of-the-art SLAM solutions optimize nonlinear least squares objectives to obtain local/globally consistent estimates of the robot state and the map. Such objectives are of the form $\frac{1}{2} \sum \mathbf{r}(\mathbf{x})^{\mathbf{2}}$, where $\mathbf{r}(\mathbf{x})$ is a nonlinear function of residuals. Example application scenarios that induce this nonlinear least squares form include visual odometry, depth measurement registration (e.g., ICP), and pose-graph optimization among others. Such objective

[6]Wherever exact differentiable realizations are not possible, we desire *as-exact-as-possible* realizations.

[7]That is, realizable as *fully* differentiable computational graphs.

functions are minimized using a succession of linear approximations $(\mathbf{r}(\mathbf{x}+\delta\mathbf{x})|_{\mathbf{x}=\mathbf{x_0}} = \mathbf{r}(\mathbf{x_0}) + \mathbf{J}(\mathbf{x_0})\delta\mathbf{x})$, using Gauss-Newton (GN) or Levenberg-Marquardt (LM) solvers. GN solvers are extremely sensitive to intialization, numerical precision, and moreover, provide no guarantees on *non-divergent* behavior. Hence most SLAM systems use LM solvers.

Trust-region methods (such as LM) are not differentiable as at each optimization step, they involve recalibration of optimizer parameters, based on a *lookahead* operation over subsequent iterates [27]. Specifically, after a new iterate is computed, LM solvers need to make a *discrete* decision between damping or undamping the linear system. Furthermore, when undamping, the iterate must be restored to its previous value. This discrete *switching* behavior of LM does not allow for gradient flow in the backward pass.

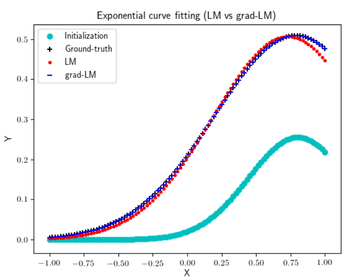

Figure 4. An example curve fitting problem, showing that ∇LM performs near-identical to LM, with the added advantage of being fully differentiable.

We propose a computationally efficient *soft reparametrization* of the damping mechanism to enable differentiability in LM solvers. Our key insight is that, if $\mathbf{r_0} = \mathbf{r}(\mathbf{x_0})^\mathbf{T}\mathbf{r}(\mathbf{x_0})$ is the norm of the error at the current iterate, and $\mathbf{r_1} = \mathbf{r}(\mathbf{x_1})^\mathbf{T}\mathbf{r}(\mathbf{x_1})$ is the norm of the error at the *lookahead* iterate, the value of $\mathbf{r_1} - \mathbf{r_0}$ determines whether to damp or to undamp. And, only when we choose to undamp, we revert to the current iterate. We define two smooth *gating* functions Q_x and Q_λ based on the generalized logistic function [36] to update the iterate and determine the next damping coefficient.

$$
\begin{aligned}
\lambda_1 = Q_\lambda(r_0, r_1) &= \lambda_{min} + \frac{\lambda_{max} - \lambda_{min}}{1 + De^{-\sigma(r_1 - r_0)}} \\
Q_x(r_0, r_1) &= x_0 + \frac{\delta x_0}{1 + e^{-(r_1 - r_0)}}
\end{aligned} \tag{1}
$$

where D and σ are tunable parameters that control the falloff [36]. Also $[\lambda_{min}, \lambda_{max}]$ is the range of values the damping function can assume. Notice that this smooth parameterization of the LM update allows the optimizer to be expressed as a fully differentiable computational graph (Fig. 3).

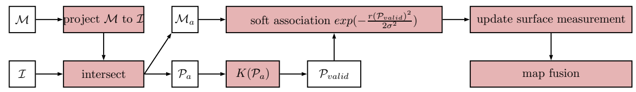

Figure 5. Computation graph for the differentiable mapping module. The uncolored boxes indicate intermediate variables, while the colored boxes indicate processing blocks. Note that the specific choice of the functions for `update surface measurement` and `map fusion` depend on the map representation used.

It must be noted that this scheme can be modified to accommodate other kinds of gating functions, such as hyperbolic curves. We however, choose the above gating functions, as they provide sufficient flexibility. A thorough treatment of the impact of the choice of gating functions on performance is left for future work.

3.4. Differentiable mapping

Another non-smooth operation in dense SLAM is map construction (*surface measurement*). For example, consider a *global* map $\mathcal{M}$ being built in the reference frame of the first image-sensor measurement I_0. When a new frame I_k arrives at time k, dense SLAM methods need to align the surface measurement being made in the live frame, with the map $\mathcal{M}$. Notwithstanding the specific choice of map representation (i.e., pointclouds, signed-distances, surfels), a generic *surface alignment* process comprises the following steps.

1. The map $\mathcal{M}$ is intersection-tested with the live frame, to determine the *active set* $\mathcal{M}_a$ of map elements, and the active set of image pixels $\mathcal{P}_a$. The remaing map elements are *clipped*.

2. Active image pixels $\mathcal{P}_a$ are checked for measurement validity (e.g., missing depth values / blurry pixels, etc.). This results in a *valid active set* of image pixls $\mathcal{P}_{valid}$

3. The set of pixels in $\mathcal{P}_{valid}$ is backprojected to 3D and compared with the map. At this stage, it must be discerned whether these pixels measure existing elements in $\mathcal{M}_a$, or if they measure a new set of elements that need to be added to the global map.

4. Once the above decision is made, these surface measurements are *fused* into the global map. The choice of the fusion mechanism is dependent on the underlying representation of each map element (points, surfels, TSDF, etc.).

The above process involves a number of differentiable yet non-smooth operations (clipping, indexing, thresholding, new/old decision, active/inactive decision, etc.). Although the above sequence of operations can be represented as a computation graph, it will not necessarily serve our purpose here since, even though (local) derivatives can be defined for operations such as clipping, indexing, thresholding, and discrete decisions, these derivatives exist only at that single point. The overall function represented by the computation graph will have undefined gradients "almost everywhere" (akin to step functions). We propose to mitigate this issue by making the functions locally *smooth*. Concretely, we propose the following corrective measures.

1. The surface measurement made by each valid pixel p in the live frame (i.e., $p \in \mathcal{P}_{valid}$) is not a function of p alone. Rather, it is the function of the pixel p and its (active/inactive) neighbours $nbd(p)$, as determined by a *kernel* $K(p, nbd(p))$.
2. When a surface measurement is transformed to the global frame, rather than using a *hard* (one-one) association between a surface measurement and a map element, we use a *soft* association to multiple map elements, in accordance with the sensor characteristics.
3. Every surface measurement is, by default, assumed to represent a new map element, which is passed to a *differentiable fusion* step (*c.f.* Sec 3.5).

The kernel $K(p, nbd(p))$ can be a discrete approximation (e.g., constant within a pixel) or can vary at the subpixel level, based on the choice of the falloff function. For faster computation and coarse gradients, we use a bilinear interpolation kernel. While bilinear interpolation is a sensible approximation for image pixels, this is often a poor choice for use in 3D *soft* associations. For forming 3D associations, we leverage characteristics of RGB-D sensors in defining the soft falloff functions. Specifically, we compute, for each point P in the live surface measurement, a set of closest candidate points in a region $exp\left(-\frac{r(P)^2}{2\sigma^2}\right)$, where $r(P)$ is the radial depth of the point from the camera ray, and σ affects the falloff region.[8]

3.5. Differentiable map fusion

The aforementioned differentiable mapping strategy, while providing us with a smooth observation model, also causes an undesirable effect: the number of map elements increases in proportion with exploration time. However, map elements should ideally increase with proportion to the *explored volume of occupied space*, rather than with exploration time. Conventional dense mapping techniques (e.g., KinectFusion [31], PointFusion [23]) employ this through *fusion* of redundant observations of the same map element. As a consequence, the recovered map has a more manageable size, but more importantly, the reconstruction quality improves greatly. While most fusion strategies are differentiable (eg. [23, 31]), they impose falloff thresholds that cause an abrupt change in gradient flow at the truncation point. We use a logistic falloff function, similar to Eq. 1, to ease gradient flow through these truncation points.

[8]This is a well-known falloff function, usually with Kinect-style depth sensors [6, 23, 32].

3.6. Differentiable ray backprojection

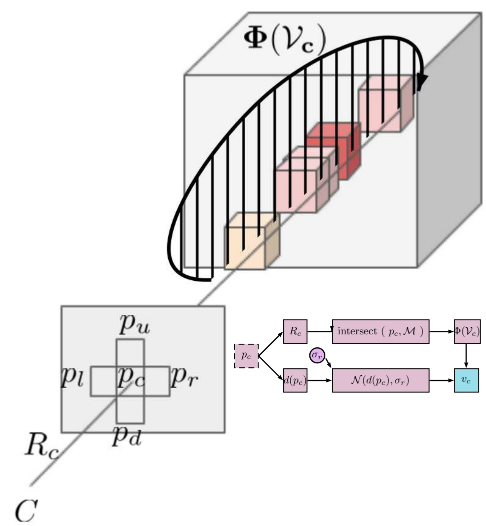

Figure 6. **Ray differentials**: Inset shows the computation graph of the ray value computation. The dashed rectangle is not differentiable, and its derivatives are approximated as shown in Eq 2

Some dense SLAM systems [31,45] perform global pose estimation by raycasting a map to a live frame. Such an operation inherently involves non-differentiable steps. First, from each pixel in the image, a ray from the camera is backprojected, and its intersection with the first map element along the direction of the ray is determined. This involves marching along the ray until a map element is found, or until we exit the bounds of reconstruction. Usual (non-differentiable) versions of ray marching use max-min acceleration schemes [33] or rely on the existence of volumetric signed distance functions [31]. Several attempts have been made to make the raycasting operation differentiable. Scene representation networks [37] proposes to predict ray marching steps using an LSTM. In other works such as DRC [42] and WS-GAN [15], the authors pool over all voxels along a ray to compute the *potential* of a ray. In this work, we make one enhancement to the ray pooling operation. We pool over all voxels along a ray, but have a Gaussian falloff defined around the depth measurement of the image pixel through which the ray passes. Further, we use finite differences to compute the derivative of the ray potential with respect to the pixel neighbourhood. We use the finite differences based ray differentials defined in Igehy *et al*. [21]. If

p_c is the image pixel that the ray R_c *pierces*, and $\mathcal{V}_c = \{v_c\}$ is the set of all voxels it pierces, then the *aggregated value* of the ray is denoted v_c (with respect to an *aggregation* function $\Phi(\psi(v_c)\ \forall v_c \in \mathcal{V}_c)$). The aggregation function simply multiplies each value $\psi(v_c)$ with the density of the Gaussian fallof at v_c, and normalizes them. Similarly v_l, v_r, v_u, and v_b are the *aggregated values* of rays emanating from the pixels to the left, right, above, and below p_c respectively. Then, the partial derivative $\frac{\partial v_c}{\partial c}$ can be approximated as

$$\frac{\partial v_c}{\partial p_c} = \begin{pmatrix} (v_r - v_l)/2 \\ (v_u - v_d)/2 \end{pmatrix} \tag{2}$$

An illustration of the ray differential computation scheme can be found in Fig. 6.

4. Case Studies: KinectFusion, PointFusion, and *ICP-SLAM*

To demonstrate the applicability of the ∇SLAM framework, we leverage the differentiable computation graphs specified in Sec 3 and compose them to realise three practical SLAM solutions. In particular, we implement differentiable versions of the *KinectFusion* [31] algorithm that constructs TSDF-based volumetric maps, the *PointFusion* [23] algorithm that constructs surfel maps, and a pointcloud-only SLAM framework that we call *ICP-SLAM*.

4.1. KinectFusion

Recall that KinectFusion [31] alternates between *tracking* and *mapping* phases. In the tracking phase, the entire up-to-date TSDF volume is raycast onto the live frame, to enable a point-to-plane ICP that aligns the live frame to the raycast model. Subsequently, in the mapping phase, surface measurements from the current frame are *fused* into the volume, using the TSDF fusion method proposed in [31]. The surface measurement is given as (*c.f.* [31])

$$\begin{gathered} sdf(p) = trunc(\|K^{-1}x\|_2^{-1}\|t-p\|_2 - depth(x)) \\ trunc(sdf) = min(1, \frac{sdf}{\mu})(sign(sdf)) \quad iff\ sdf\ \geq -\mu \end{gathered} \tag{3}$$

Here, p is the location of a voxel in the camera frame, and $x = \lfloor \pi(Kp) \rfloor$ is the live frame pixel to which p projects to. μ is a parameter that determines the threshold beyond which a surface measurement is invalid. However, we note that the *floor* operator is non-differentiable "almost everywhere". Also, the truncation operator, while differentiable within a distance of μ from the surface, is abruptly truncated, which hinders gradient flow . Instead, we again use a generalized logistic function [36] to create a smooth truncation, which provides better-behaved gradients at the truncation boundary. The other steps involved here, such as raycasting, ICP, etc. are already differentiable in the ∇SLAM framework (*c.f.* Sec 3). Fusion of surface measurements is perfomed using the same approach as in [31] (weighted averaging of TSDFs).

4.2. PointFusion

As a second example, we implement PointFusion [23], which incrementally fuses surface measurements to obtain a global surfel map. Surfel maps compare favourably to volumetric maps due to their reduced memory usage.[9] We closely follow our differentiable mapping formulation (*c.f.* Sec 3.4) and use surfels as map elements. We adopt the fusion rules from [23] to perform map fusion.

4.3. *ICP-SLAM*

As a baseline example, we implement a simple pointcloud based SLAM technique, which uses ICP to incrementally register pointclouds to a global pointcloud set. In particular, we implement two systems. The first one aligns every pair of consecutive incoming frames, to obtain an odometry estimate (also referred to as *frame-to-frame alignment* or ICP-Odometry). The second variant performs what we call *frame-to-model alignment* (ICP-SLAM). That is, each incoming frame is aligned (using ICP) with a pointcloud containing the entire set of points observed thus far.

5. Experiments and results

5.1. Differentiable optimization

In Sec 3.3, we introduced two generalized logistic functions Q_λ and Q_x to compute the damping functions as well as the subsequent iterates. We conduct multiple experiments to verify the impact of this approximation on the performance (convergence speed, quality of solution) of nonlinear least squares solvers.

We first design a test suite of nonlinear curve fitting problems (inspiration from [5]), to measure the performance of ∇LM to its non-differentiable counterpart. We consider three nonlinear functions, *viz. exponential*, *sine*, and *sinc*, each with three parameters a, t, and w.

$$\begin{aligned} f(x) &= a\ exp\left(-\frac{(x-t)^2}{2w^2}\right) \\ f(x) &= sin(ax + tx + w) \\ f(x) &= sinc(ax + tx + w) \end{aligned} \tag{4}$$

For each of these functions, we uniformly sample the parameters $p = \{a, t, w\}$ to create a suite of ground-truth curves, and uniformly sample an initial guess $p_0 = \{a_0, t_0, w_0\}$ in the interval $[-6, 6]$. We sample 100 problem instances for each of the three functions. We run a variety of optimizers (such as gradient descent (GD), Gauss-Newton

[9]On the flipside, surfel-based algorithms are harder to parallelize compared to volumetric fusion.

T_{max} = 10 iters	*Exponential*				*Sine*				*Sinc*			
	GD	**GN**	**LM**	**∇LM**	**GD**	**GN**	**LM**	**∇LM**	**GD**	**GN**	**LM**	**∇LM**
$\|a_{pred} - a_{gt}\|^2$	**0.422**	0.483	0.483	0.483	0.379	**0.341**	0.342	0.342	2.929	0.304	0.304	**0.304**
$\|t_{pred} - t_{gt}\|^2$	0.606	**0.50**	0.550	0.550	**0.222**	0.359	0.360	0.360	3.024	0.304	0.304	**0.040**
$\|w_{pred} - w_{gt}\|^2$	1.268	0.667	0.075	**0.075**	1.215	**0.080**	0.084	0.085	0.462	$\mathbf{10^{-7}}$	0.023	10^{-4}
$\|f(x)_{pred} - f(x)_{gt}\|^2$	0.716	0.160	0.163	**0.160**	0.666	0.148	0.152	**0.148**	0.700	$\mathbf{5 \times 10^{-8}}$	0.005	4×10^{-5}
T_{max} = 50 iters												
$\|a_{pred} - a_{gt}\|^2$	0.365	0.275	**0.231**	0.275	0.486	**0.429**	0.434	0.434	3.329	0.380	0.380	**0.380**
$\|t_{pred} - t_{gt}\|^2$	0.263	0.219	0.231	**0.218**	0.519	**0.455**	0.459	0.460	2.739	0.380	0.380	**0.380**
$\|w_{pred} - w_{gt}\|^2$	1.220	0.205	**0.007**	0.369	1.327	0.273	**0.376**	0.383	0.383	$\mathbf{2 \times 10^{-7}}$	0.202	4×10^{-5}
$\|f(x)_{pred} - f(x)_{gt}\|^2$	0.669	0.083	**0.004**	0.078	0.673	0.153	0.153	**0.151**	0.795	$\mathbf{2 \times 10^{-7}}$	0.005	3×10^{-5}
T_{max} = 100 iters												
$\|a_{pred} - a_{gt}\|^2$	**0.431**	0.475	0.480	0.487	0.486	**0.429**	0.434	0.434	2.903	0.196	0.196	**0.196**
$\|t_{pred} - t_{gt}\|^2$	0.466	**0.311**	0.378	0.323	0.519	**0.455**	0.459	0.460	2.847	0.196	0.196	**0.196**
$\|w_{pred} - w_{gt}\|^2$	1.140	0.364	0.066	**0.065**	1.327	**0.273**	0.376	0.382	0.601	$\mathbf{10^{-7}}$	0.026	9×10^{-5}
$\|f(x)_{pred} - f(x)_{gt}\|^2$	0.662	0.243	**0.162**	0.230	0.673	0.153	0.153	**0.151**	0.707	$\mathbf{6 \times 10^{-8}}$	0.005	4×10^{-5}

Table 1. **∇LM** performs quite similarly to its non-differentiable counterpart, on a variety of non-linear functions, and at various stages of optimization. Here, **GD**, **GN**, and **LM** refer to gradient descent, Gauss-Newton, and Levenberg-Marquardt optimizers respectively.

(GN), LM, and ∇LM) for a maximum of $10, 50$, and 100 iterations. We compute the mean squared error in *parameter space* (independently for each parameter a, t, w) as well as in *function space* (i.e., $\|f(x)_{pred} - f(x)_{gt}\|^2$. Note that these two errors are not necessarily linearly related, as the interaction between the parameters and the function variables are highly nonlinear. The results are presented in Table 4.3. It can be seen that ∇LM performs near-identically to LM.

5.2. Comparitive analysis of case studies

In Sec 4, we implemented KinectFusion [31], PointFusion [23], and ICP-SLAM as differentiable computational graphs. Here, we present an analysis of how each of the approaches compare to their non-differentiable counterparts. Table 2 shows the trajectory tracking performance of the non-differentiable and differentiable (∇) versions of ICP-Odometry, ICP-SLAM, and PointFusion. We observe no virtual change in performance when utilizing the differentiable mapping modules and ∇LM for optimization. This is computed over split subsets of the `living_room_traj0` sequence.

We also evaluate the reconstruction quality of ∇-KinectFusion with that of Kintinuous [45]. On a subsection of the `living_room_traj0` sequence of the ICL-NUIM [18] benchmark, the surface reconstruction quality of Kintinuous is 18.625, while that of differentiable KinectFusion is 21.301 (better). However, this quantity is misleading, as Kintinuous only retains a subset of high confidence points in the extracted mesh, while our differentiable KinectFusion outputs (see Fig. 8) contain a few noisy artifacts, due to our smooth truncation functions.

Method	ATE	RPE
ICP-Odometry (non-differentiable)	0.029	0.0318
∇ICP-Odometry	**0.01664**	**0.0237**
ICP-SLAM (non-differentiable)	0.0282	0.0294
∇ICP-SLAM	**0.01660**	**0.0204**
PointFusion (non-differentiable)	**0.0071**	**0.0099**
∇PointFusion	0.0072	0.0101
KinectFusion (non-differentiable)	**0.013**	**0.019**
∇KinectFusion	0.016	0.021

Table 2. **Performance of ∇SLAM**. The differentiable counterparts perform nearly similar to their non-differentiable counterparts (ATE: Absolute Trajectory Error, RPE: Relative Pose Error).

5.3. Qualitative results

∇SLAM works out of the box on multiple other RGB-D datasets. Specifically, we present qualitative results of running our differentiable SLAM systems on RGB-D sequences from the TUM RGB-D dataset [39], ScanNet [7], as well as on an in-house sequence captured from an Intel RealSense D435 camera.

Fig. 9- 11 show qualitative results obtained by running ∇SLAM on a variety of sequences from the TUM RGB-D benchmark (Fig. 9), ScanNet (Fig. [7]), and an in-house sequence (Fig. 11). These differentiable SLAM systems all execute fully on the GPU, and are capable of computing gradients with respect to *any* intermediate variable (Eg. camera poses, pixel intensities/depths, optimization parameters, camera intrinsics, etc.).

5.4. Analysis of Gradients

The computational graph approach of ∇SLAM allows us to recover meaningful gradients of 2D (or 2.5D) measurements with respect to a 3D surface reconstruction. In Fig. 12, the top row shows an RGB-D image differentiably transformed—using ∇SLAM—into a (noisy) TSDF sur-

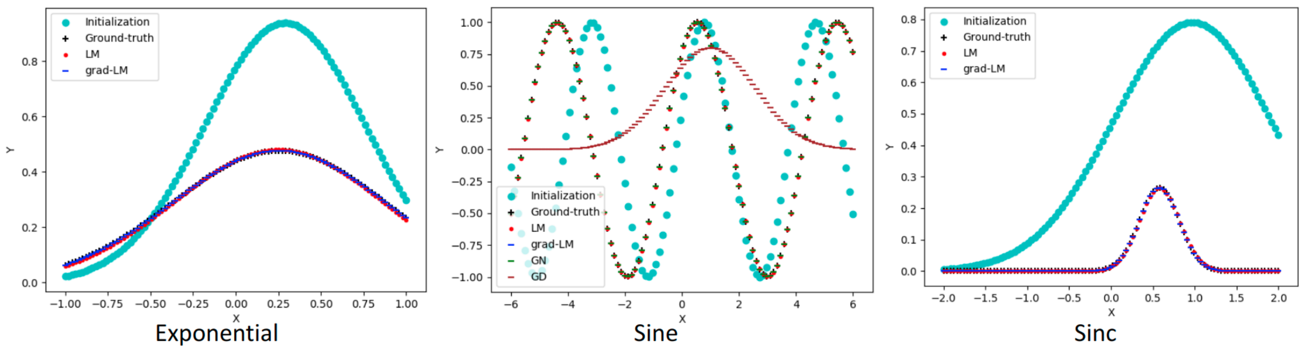

Figure 7. ∇**LM** performs comparably to LM optimizers. In this figure, we show example curve fitting problems from the test suite.

Figure 8. **Qualitative results**: On the **`living room lr kt0`** sequence of the ICL-NUIM dataset [18]. The reconstructions are near-identical to their non-differentiable counterparts. However, distinct from classic SLAM approaches, these reconstructions allow for gradients to flow from a 3D map element all the way to the entire set of pixel-space measurements of that element.

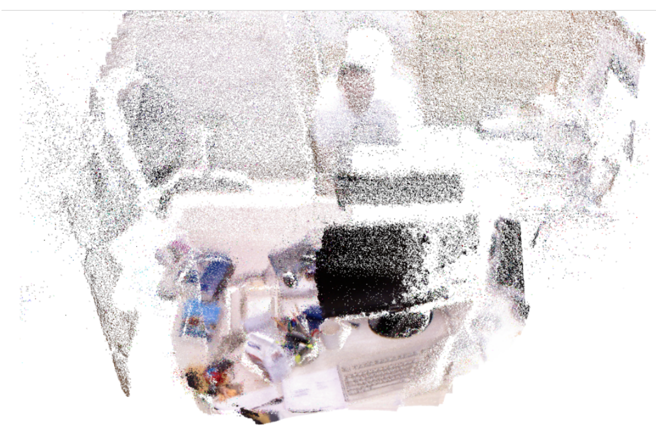

Figure 9. Reconstruction obtained upon running the differentiable ICP-Odometry pipeline on a subsection of the `rgbd_dataset_freiburg1_xyz` sequence.

face measurement, and then compared to a more precise global TSDF map. Elementwise comparision of aligned volumes gives us a reconstruction error, whose gradients are backpropagated through to the input depthmap using the computational graph maintained by ∇SLAM (and visualized in the depth image space). In the second row, we intentionally introduce an occluder that masks out a small (40×40) region in the RGB-D image, thereby introducing a reconstruction artifact. Computing the volumetric error between the global and local occluded TSDF volumes and inspecting the gradients with respect to the input indicates the per pixel contribution of the occluding surface to the volumetric error. *Thus, ∇SLAM provides a rich interpretation of the computed gradients: they denote the contribution of each pixel towards the eventual 3D reconstruction.*

6. Conclusion

We introduce ∇SLAM, a differentiable computational graph framework that enables gradient-based learning for a large set of localization and mapping based tasks, by providing explicit gradients with respect to the input image and depth maps. We demonstrate a diverse set of case studies, and showcase how the gradients propogate throughout the tracking, mapping, and fusion stages. Future efforts will enable ∇SLAM to be directly plugged into and optimized in conjunction with downstream tasks. ∇SLAM can also enable a variety of self-supervised learning applications, as any gradient-based learning architecture can now be equipped with a sense of *spatial understanding*.

References

[1] Brandon Amos and J. Zico Kolter. OptNet: Differentiable optimization as a layer in neural networks. In *ICML*, 2017. 3

[2] Michael Bloesch, Jan Czarnowski, Ronald Clark, Stefan Leutenegger, and Andrew J Davison. Codeslam—learning a compact, optimisable representation for dense visual slam. In *CVPR*, 2018. 2

Figure 10. **Qualitative results** on sequences from the ScanNet [7] dataset. Owing to GPU memory constraints, we use each of the differentiable SLAM systems (∇KinectFusion, ∇PointFusion, and ∇ICP-SLAM) to reconstruct parts of the scene. We also show outputs from BundleFusion [8] for reference.

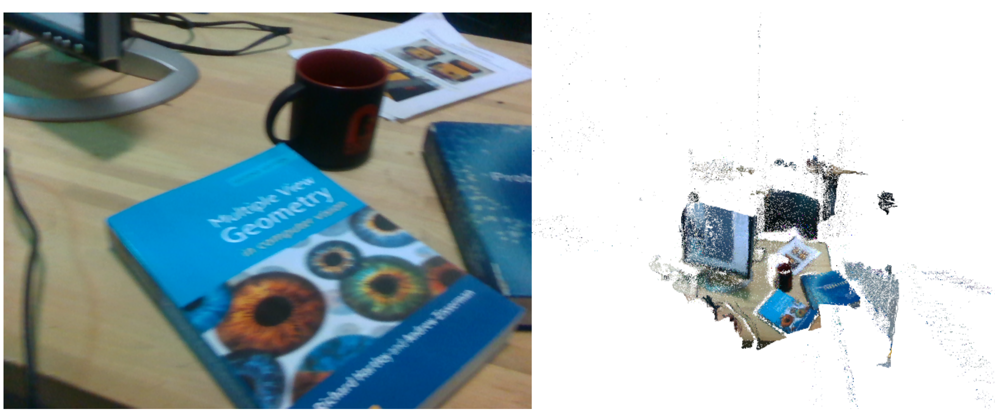

Figure 11. **In-house sequence** collected from an Intel RealSense D435 camera. The reconstruction (right) is obtained by running ∇PointFusion. Note that we do not perform any noise removal. Differentiable noise filtering is left for future work.

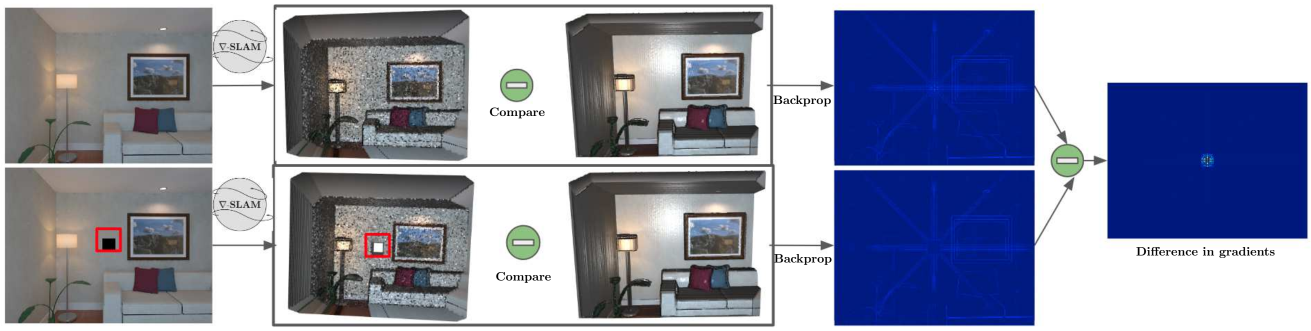

Figure 12. **Analysis of gradients**: ∇SLAM enables gradients to flow all the way back to the input images. *Top*: An RGB-D image pair (depth not shown) is passed through ∇SLAM, and reconstruction error is computed using a precise fused map. Backpropagation passes these gradients all the way back to the depth map (blue map). *Bottom*: An explicit occluder added to the center of the RGB-D pair. This occluder distorts the construction by creating a gaping hole through it. But, using the backpropagated gradients, one can identify the set of image/depthmap pixels that result in a particular area to be reconstructed imperfectly.

[3] Bert De Brabandere, Wouter Van Gansbeke, Davy Neven, Marc Proesmans, and Luc Van Gool. End-to-end lane detection through differentiable least-squares fitting. *CoRR*, abs/1902.00293, 2019. 2

[4] Devendra Singh Chaplot, Emilio Parisotto, and Ruslan Salakhutdinov. Active neural localization. In *International Conference on Learning Representations*, 2018. 2

[5] Ronald Clark, Michael Bloesch, Jan Czarnowski, Stefan Leutenegger, and Andrew J Davison. Ls-net: Learning to solve nonlinear least squares for monocular stereo. *ECCV*, 2018. 3, 6

[6] Brian Curless and Marc Levoy. A volumetric method for building complex models from range images. 1996. 5

[7] Angela Dai, Angel X. Chang, Manolis Savva, Maciej Halber, Thomas Funkhouser, and Matthias Nießner. Scannet: Richly-annotated 3d reconstructions of indoor scenes. In *Proc. Computer Vision and Pattern Recognition (CVPR), IEEE*, 2017. 7, 9, 12

[8] Angela Dai, Matthias Nießner, Michael Zollöfer, Shahram Izadi, and Christian Theobalt. Bundlefusion: Real-time globally consistent 3d reconstruction using on-the-fly surface re-integration. *ACM Transactions on Graphics 2017 (TOG)*, 2017. 9

[9] Andrew J. Davison. Futuremapping: The computational structure of spatial AI systems. *CoRR*, 2018. 1

[10] D. Ponsa E. Rublee E. Riba, D. Mishkin and G. Bradski. Kornia: an open source differentiable computer vision library for pytorch. In *Winter Conference on Applications of Computer Vision*, 2019. 2

[11] J. Engel, T. Schöps, and D. Cremers. LSD-SLAM: Large-scale direct monocular SLAM. In *ECCV*, 2014. 2

[12] Ravi Garg, Vijay Kumar BG, Gustavo Carneiro, and Ian Reid. Unsupervised cnn for single view depth estimation: Geometry to the rescue. In *ECCV*, 2016. 2

[13] S. K. Gottipati, K. Seo, D. Bhatt, V. Mai, K. Murthy, and L. Paull. Deep active localization. *IEEE Robotics and Automation Letters*, 4(4):4394–4401, Oct 2019. 2

[14] Edward Grefenstette, Brandon Amos, Denis Yarats, Phu Mon Htut, Artem Molchanov, Franziska Meier, Douwe Kiela, Kyunghyun Cho, and Soumith Chintala. Generalized inner loop meta-learning. *arXiv preprint arXiv:1910.01727*, 2019. 3

[15] JunYoung Gwak, Christopher B Choy, Manmohan Chandraker, Animesh Garg, and Silvio Savarese. Weakly supervised 3d reconstruction with adversarial constraint. In *2017 International Conference on 3D Vision (3DV)*, 2017. 5

[16] Lei Han, Mengqi Ji, Lu Fang, and Matthias Nießner. Regnet: Learning the optimization of direct image-to-image pose registration. *CoRR*, abs/1812.10212, 2018. 3

[17] Ankur Handa, Michael Bloesch, Viorica Pătrăucean, Simon Stent, John McCormac, and Andrew Davison. gvnn: Neural network library for geometric computer vision. In *ECCV Workshop on Geometry Meets Deep Learning*, 2016. 2

[18] A. Handa, T. Whelan, J.B. McDonald, and A.J. Davison. A benchmark for RGB-D visual odometry, 3D reconstruction and SLAM. In *ICRA*, 2014. 7, 8

[19] Geoffrey Hinton, Li Deng, Dong Yu, George Dahl, Abdel-rahman Mohamed, Navdeep Jaitly, Andrew Senior, Vincent Vanhoucke, Patrick Nguyen, Brian Kingsbury, et al. Deep neural networks for acoustic modeling in speech recognition. *IEEE Signal processing magazine*, 2012. 1

[20] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 9:1735–80, 12 1997. 3

[21] Homan Igehy. Tracing ray differentials. In *Proceedings of the 26th annual conference on Computer graphics and interactive techniques*, pages 179–186. ACM Press/Addison-Wesley Publishing Co., 1999. 5

[22] Max Jaderberg, Karen Simonyan, Andrew Zisserman, et al. Spatial transformer networks. In *Neurips*, 2015. 2

[23] Maik Keller, Damien Lefloch, Martin Lambers, Shahram Izadi, Tim Weyrich, and Andreas Kolb. Real-time 3d reconstruction in dynamic scenes using point-based fusion. In *2013*, 2013. 2, 3, 5, 6, 7, 11

[24] Alex Kendall, Matthew Grimes, and Roberto Cipolla. Posenet: A convolutional network for real-time 6-dof camera relocalization. In *ICCV*, 2015. 2

[25] Christian Kerl, Jürgen Sturm, and Daniel Cremers. Robust odometry estimation for rgb-d cameras. In *ICRA*, 2013. 2, 11

[26] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In *NeurIPS*, 2012. 1

[27] Michael Lampton. Damping–undamping strategies for the levenberg–marquardt nonlinear least-squares method. *Computers in Physics*, 1997. 4

[28] Ruihao Li, Sen Wang, Zhiqiang Long, and Dongbing Gu. Undeepvo: Monocular visual odometry through unsupervised deep learning. In *ICRA*, 2018. 2

[29] Bruce D Lucas, Takeo Kanade, et al. An iterative image registration technique with an application to stereo vision. 1981. 2

[30] B. Mu, S. Liu, L. Paull, J. Leonard, and J. P. How. Slam with objects using a nonparametric pose graph. In *2016 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 4602–4609, Oct 2016. 2

[31] Richard A Newcombe, Shahram Izadi, Otmar Hilliges, David Molyneaux, David Kim, Andrew J Davison, Pushmeet Kohli, Jamie Shotton, Steve Hodges, and Andrew W Fitzgibbon. Kinectfusion: Real-time dense surface mapping and tracking. In *ISMAR*, 2011. 2, 3, 5, 6, 7, 11, 12

[32] Chuong V Nguyen, Shahram Izadi, and David Lovell. Modeling kinect sensor noise for improved 3d reconstruction and tracking. In *2012 second international conference on 3D imaging, modeling, processing, visualization & transmission*. IEEE, 2012. 5

[33] Steven Parker, Peter Shirley, Yarden Livnat, Charles Hansen, and P-P Sloan. Interactive ray tracing for isosurface rendering. In *Proceedings Visualization'98 (Cat. No. 98CB36276)*, pages 233–238. IEEE, 1998. 5

[34] P. Parkhiya, R. Khawad, J. K. Murthy, B. Bhowmick, and K. M. Krishna. Constructing category-specific models for monocular object-slam. In *2018 IEEE International Conference on Robotics and Automation (ICRA)*, pages 1–9, May 2018. 2

[35] Adam Paszke, Sam Gross, Soumith Chintala, Gregory Chanan, Edward Yang, Zachary DeVito, Zeming Lin, Alban Desmaison, Luca Antiga, and Adam Lerer. Automatic differentiation in pytorch. 2017. 11, 12

[36] FJ Richards. A flexible growth function for empirical use. *Journal of experimental Botany*, 1959. 4, 6

[37] Vincent Sitzmann, Michael Zollhöfer, and Gordon Wetzstein. Scene representation networks: Continuous 3d-structure-aware neural scene representations. In *Advances in Neural Information Processing Systems*, 2019. 5

[38] Frank Steinbrücker, Jürgen Sturm, and Daniel Cremers. Real-time visual odometry from dense rgb-d images. In *ICCV Workshops*, 2011. 2

[39] J. Sturm, N. Engelhard, F. Endres, W. Burgard, and D. Cremers. A benchmark for the evaluation of rgb-d slam systems. In *Proc. of the International Conference on Intelligent Robot Systems (IROS)*, 2012. 7, 12

[40] Chengzhou Tang and Ping Tan. Ba-net: Dense bundle adjustment network. *ICLR*, 2019. 3

[41] Tombari F. Laina I. Navab N. Tateno, K. Cnn-slam: Real-time dense monocular slam with learned depth prediction. In *CVPR*, 2017. 2

[42] Shubham Tulsiani, Tinghui Zhou, Alexei A. Efros, and Jitendra Malik. Multi-view supervision for single-view reconstruction via differentiable ray consistency. In *CVPR*, 2017. 5

[43] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *NeurIPS*, 2017. 1

[44] Lukas von Stumberg, Patrick Wenzel, Qadeer Khan, and Daniel Cremers. Gn-net: The gauss-newton loss for deep direct SLAM. *CoRR*, abs/1904.11932, 2019. 3

[45] Thomas Whelan, Michael Kaess, Hordur Johannsson, Maurice Fallon, John J Leonard, and John McDonald. Real-time large-scale dense rgb-d slam with volumetric fusion. *The International Journal of Robotics Research*, 34(4-5):598–626, 2015. 2, 5, 7

[46] S. Yang and S. Scherer. Cubeslam: Monocular 3-d object slam. *IEEE Transactions on Robotics*, 35(4):925–938, Aug 2019. 2

[47] Shuaifeng Zhi, Michael Bloesch, Stefan Leutenegger, and Andrew J Davison. Scenecode: Monocular dense semantic reconstruction using learned encoded scene representations. In *CVPR*, 2019. 2

[48] Huizhong Zhou, Benjamin Ummenhofer, and Thomas Brox. Deeptam: Deep tracking and mapping. In *ECCV*, 2018. 2

[49] Tinghui Zhou, Matthew Brown, Noah Snavely, and David G Lowe. Unsupervised learning of depth and ego-motion from video. In *CVPR*, 2017. 2

A. ∇SLAM: Library

In this paper, we demonstrated that classical dense SLAM systems such as KinectFusion [31], PointFusion [23], and ICP-SLAM can all be realized as differentiable computations. However, the set of differentiable modules introduced herein can be used to construct several newer differentiable SLAM systems. To this end, we intend to make the ∇SLAM framework publicly available as open-source software (anticipated release date: late January, 2020).

∇SLAM is built on top of PyTorch [35], a reverse-mode automatic differentiation library that supports computation over multi-dimensional arrays (often misnomered tensors). At the time of writing this article, ∇SLAM supports the following functionality[10]:

1. Non-linear least squares optimization
2. Depth-based perspective warping (dense visual odometry [25])
3. Point-to-plane ICP
4. Raycasting
5. TSDF volumetric fusion

[10] All of these operations are performed fully differentiably.

6. PointFusion (surfel map building)

7. ICP-SLAM

8. Boilerplate operations (Lie algebraic utilities, differentiable vertex and normal map computation, etc.)

∇SLAM is intended to be an *out-of-the-box* PyTorch-based SLAM framework. Currently, it interfaces with popular datasets such as ScanNet [7], TUM RGB-D benchmark [39], etc. By the time of release, we plan on extending functionality to other popular datasets, and also focus on real-time performance. Currently, low resolution reconstructions (for example, a $128 \times 128 \times 128$ TSDF volume runs at about $10Hz$ on a medium-end laptop GPU (NVIDIA GeForce 1060).

For more details on release timelines, and for more visualizations/results, one can visit this webpage.

B. Frequently asked questions (FAQ)

1. **Q:** *So, ∇SLAM is just classical dense SLAM, implemented using an autograd-compatible language/library?*
   A: Yes and no. Technically, while it is possible to “simply" implement dense SLAM in an autograd-compatible library (eg. PyTorch [35]), in such a case the obtained gradients would not be meaningful enough, to be used in a gradient-based learning pipeline. We believe that ∇SLAM addresses many such problems (of the gradients being zero “almost everywhere", akin to impulse functions).

2. **Q:** *The paper paints a rosy side of ∇SLAM. What are some of the shortcomings of the framework?*
   A: Unrolling each computation in dense SLAM as a graph requires an enormous amount of memory. For example, running a differentiable KinectFusion [31] algorithm using a coarse voxel resolution $128 \times 128 \times 128$ ends up requiring $6GB$ of GPU memory on average. This severely restricts the size of scenes that can be reconstructed in this framework. That is one of the primary concerns we are tackling at the moment. Another aspect of ∇SLAM we are improving upon is to add more robust (differentiable) filters into several stages of the pipeline, such as ICP, photometric warping, etc. We are also working on getting in M-estimators into the optimization routine.

3. **Q:** *What is the application of such a system?* **A:** We envisage a plethora of applications for a differentiable SLAM system, ranging from enabling spatially-grounded learning, to self-supervision, to task-oriented learning and beyond. We believe that ∇SLAM greatly benefits by following the same modular structure as conventional SLAM systems. This could potentially allow localized learning in only submodules of a SLAM system that actually need to be learnt.

C. Acknowledgements

The authors thank Gunshi Gupta for helping out with some of the ∇LM experiments. KM would like to thank Ankur Handa for providing valuable feedback/advice on this work. KM also thanks Zeeshan Zia, Ronald Clark, and Sajad Saeedi, who participated in the initial brainstorming sessions. KM and GI acknowledge timely help from Aaditya Saraiya, Parv Parkhiya, Akshit Gandhi, Shubham Garg, all from CMU, who helped capture a live sequence that features in the ∇SLAM video. During the early stages of the project, KM benefitted from participating in brainstorming sessions with Tejas Khot and Gautham Swaminathan. The authors acknowledge the impact of the feedback from colleagues at the Robotics and Embodied AI Lab (REAL), Université de Montréal on this work; in particular, Gunshi Gupta, Bhairav Mehta, and Mark Van der Merwe.