Multi-level mapping: Real-time dense monocular SLAM

Multi-Level Mapping: Real-time Dense Monocular SLAM

W. Nicholas Greene[1], Kyel Ok[1], Peter Lommel[2], and Nicholas Roy[1]

***Abstract*— We present a method for Simultaneous Localization and Mapping (SLAM) using a monocular camera that is capable of reconstructing dense 3D geometry online without the aid of a graphics processing unit (GPU).**

Our key contribution is a multi-resolution depth estimation and spatial smoothing process that exploits the correlation between low-texture image regions and simple planar structure to adaptively scale the complexity of the generated keyframe depthmaps to the texture of the input imagery. High-texture image regions are represented at higher resolutions to capture fine detail, while low-texture regions are represented at coarser resolutions for smooth surfaces. The computational savings enabled by this approach allow for significantly increased reconstruction density and quality when compared to the state-of-the-art. The increased depthmap density also improves tracking performance as more constraints can contribute to the pose estimation. A video of experimental results is available at `http://groups.csail.mit.edu/rrg/multi_level_mapping`.

Fig. 1: Our multi-level depth estimation and spatial regularization process enables dense point cloud maps to be generated online without GPU acceleration. This point cloud was generated in real-time from approximately two minutes of 30 Hz video around a laboratory workspace.

I. Introduction

Cameras are powerful sensors for robotic navigation because they provide rich environment information (color, shape, texture, etc.) at high resolution and long ranges, while being lightweight, low-power, and inexpensive. Exploiting such sensor data for navigation tasks typically falls into the realm of monocular SLAM, where both the robot's pose and a map of the environment are estimated concurrently from the imagery produced by a single camera mounted on the robot.

Traditionally, real-time methods capable of operating at the camera frame rate have relied on computing sparse interest points or features (e.g. [1]–[3]) in each camera frame, matching these features across frames, and then using these feature tracks to estimate the camera trajectory and the 3D positions of the features [4]–[7]. While the tracking performance of such algorithms is impressive, the sparse point-based maps generated have limited utility for robotic navigation because they do not sufficiently distinguish between free and occupied space and therefore cannot be used for motion planning.

Recently, however, advances have been made in real-time, dense and semi-dense methods that are able to construct high-quality 3D scene models by leveraging many small-baseline stereo comparisons using raw pixel intensities rather than a small number of large-baseline stereo comparisons with robust feature descriptors [8]–[14]. All these methods, however, struggle in low-texture environments and require expensive global regularization procedures (often requiring GPU acceleration) to smoothly interpolate over these regions.

Our approach is guided by two observations about the low-texture image regions that are particularly troublesome for visual SLAM systems: (1) in the absence of high-frequency information such as sharp image gradients, depth estimation is typically unreliable and (2) in man-made environments, low-texture regions are often correlated with simple or planar surfaces. These observations suggest that attempting to estimate depth in low-texture regions at native image resolution is misguided: the estimation process is inherently error-prone and the resulting (noisy) depths must be smoothed out using expensive spatial regularization. Useful information may still be extracted from these regions, however, by leveraging lower-frequency features or gradients.

Motivated by these observations, our method uses variable-resolution quadtrees to represent keyframe images, where we estimate the depth of each leaf in the quadtree, rather than each pixel in the image. The quadtree representation of the keyframe allows us to model different image regions at a scale appropriate for the available texture: high-texture regions will be represented at the finest scale, while low-texture regions will be represented at coarser scales. This approach has a number of advantages: (1) the computational budget is more efficiently distributed among the image pixels, (2) low-texture regions are represented at a level of detail appropriate to their information content, (3) we can apply variational regularization techniques without

[1]Computer Science and Artificial Intelligence Laboratory (CSAIL), Massachusetts Institute of Technology, Cambridge, MA 02139, USA {wng,kyelok,nickroy}@csail.mit.edu

[2] The Charles Stark Draper Laboratory, Cambridge, MA 02139, USA plommel@draper.com

This work was supported by a research fellowship from the Charles Stark Draper Laboratory. Their support is greatly acknowledged.

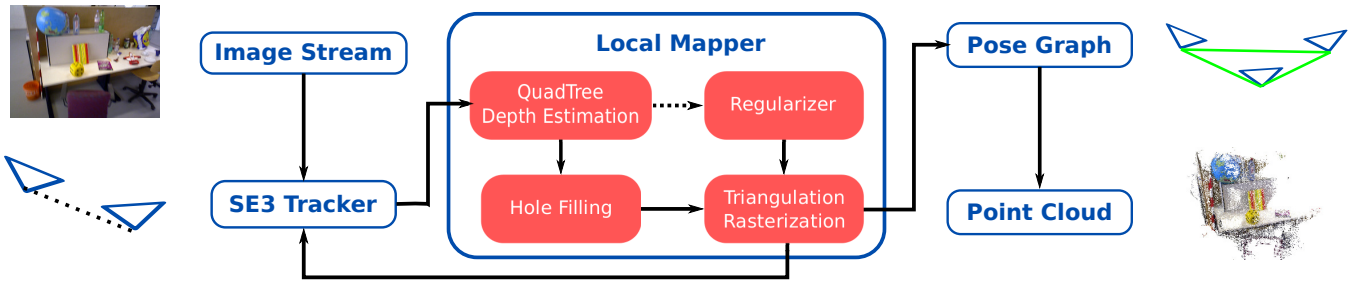


Fig. 2: *Monocular SLAM Pipeline*: Incoming images are first tracked in $\mathbb{SE}(3)$ relative to the current keyframe using dense, direct image alignment. Tracked frames are then passed to the Local Mapper, which estimates a quadtree-based, multi-resolution inverse depthmap using many short-baseline stereo computations. Holes in the depthmap are then filled before being interpolated back into the native image resolution using a simple software rasterization procedure. When a keyframe is finished, it is passed to a variational regularizer which removes outliers and smooths away noise. The keyframe is then inserted into a pose-graph defined on $\mathrm{Sim}(3)$ and is incrementally aligned to the other keyframes. Finally, the depthmaps are projected into 3D and visualized as colored point clouds.

GPU acceleration, and (4) low-frequency image features can be exploited when high-frequency information is absent.

The key technical challenge is capturing the spatial correlations between the different scales of the quadtree leaf representation to infer a smooth estimate of the inverse depthmap at full resolution. The contribution of this paper is to first show how the multiscale representation can be inferred and then show how the multiscale representation enables smoothing across scales using a primal-dual optimization that is then used to infer a dense depthmap at native resolution. Our approach results in both denser and more accurate keyframes than the state of the art direct methods such as LSD-SLAM [14] in comparable time.

II. Related Work

While 3D reconstruction from 2D images is a classical problem in computer vision (typically dubbed multi-view stereo or Structure from Motion in the literature), the first monocular SLAM system to operate in real-time is commonly attributed to [6], which used an extended Kalman filter (EKF) to recursively estimate the camera pose and sparse feature positions. While an important milestone, the cubic complexity of the EKF constrains the state space dimension and caps the number of features that can be present at any time in the filter.

The Parallel Tracking and Mapping (PTAM) algorithm developed by [7] sidesteps this constraint by splitting the tracking and mapping computations into separate threads that run in parallel at different rates. This allows for tracking to occur at frame rate with only the relevant features in view, while a separate thread performs costly least-squares optimization to align all the features present in the map at a lower rate.

Similarly innovative, the Dense Tracking and Mapping (DTAM) algorithm of [8] combines this parallel approach with the horsepower of the GPU and variational regularization inspired by [15] to achieve impressive real-time results with both dense tracking and dense mapping. Similar solutions utilizing the GPU and variational methods for regularization were also developed by [9]–[11].

Multi-resolution depth estimation techniques also have precedence in the multi-view stereo literature, with several algorithms developed in the past 15 years proposing approaches that leverage multiple image scales [16]–[18]. SLAM solutions leveraging hierarchical maps or multi-resolution features have also been explored [19]–[21].

Our approach is most similar, however, to the so-called "direct" methods of [12]–[14], which can be thought of as a hybrid between the aforementioned sparse and dense techniques. The Semi-direct Visual Odometry (SVO) algorithm of [12] estimates camera pose and 3D points using the raw pixel intensity patches of a sparse set of keypoints rather than expensive feature descriptors (hence the "direct" label) and is able to run at nearly 300 Hz on a commodity desktop computer without a GPU. The Large-Scale Direct-SLAM (LSD-SLAM) algorithm [13], [14] also uses raw pixel intensities, but leverages all pixels with sufficient image gradient instead of just a sparse subset, allowing for large, semi-dense point cloud maps to be generated at frame-rate. Our work builds on top of [14] and extends the mapping component to produce accurate, dense point clouds while maintaining real-time operation.

III. Problem Formulation

A. Notation

Let $I_t : \Omega \to \mathbb{R}$ represent an $m \times n$ image taken from a camera at time index t, where $\Omega \subset \mathbb{R}^2$ represents the image pixel domain. We denote the pose of the camera at time t with respect to keyframe k (up to the unobservable global scale factor) by

$$\mathbf{T}_t^k = \begin{bmatrix} \mathbf{R} & \mathbf{t} \\ 0 & 1 \end{bmatrix} \in \mathbb{SE}(3), \tag{1}$$

where $\mathbf{R} \in \mathbb{SO}(3)$ and $\mathbf{t} \in \mathbb{R}^3$. We let $\bar{\mathbf{x}} = \begin{bmatrix} \mathbf{x}^T & 1 \end{bmatrix}^T$ represent the homogeneous coordinates of $\mathbf{x}$ such that a point $\mathbf{p}_t \in \mathbb{R}^3$ in frame t can be transformed into frame k by $\bar{\mathbf{p}}_k = \mathbf{T}_t^k \bar{\mathbf{p}}_t$.

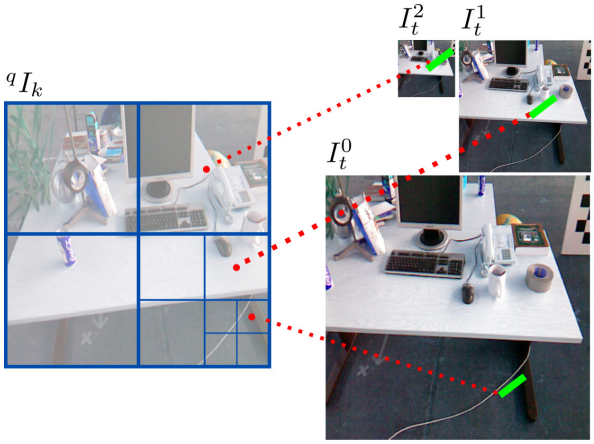


Fig. 3: *Multi-level Depth Estimation*: We perform stereopsis at the image scale appropriate for the available texture. For each keyframe I_k we compute a quadtree-compressed representation ^qI_k based on the pixel intensities and define an inverse depthmap over the leaf nodes (left). For each newly tracked frame I_t, we compute its power-of-two image pyramid ^LI_t and perform small-baseline stereo computations between nodes in ^qI_k and pixels in ^LI_t at the image scale corresponding to the given node (right). The green lines represents the epipolar search regions for each stereo computation.

We let the matrix $\mathbf{K} \in \mathbb{R}^{3\times3}$ represent the camera intrinsic parameters and define the perspective projection function as $\pi\left(\begin{bmatrix}x & y & z\end{bmatrix}^T\right) = \begin{bmatrix}x/z & y/z\end{bmatrix}^T$. The projection of point $\mathbf{p}_k \in \mathbb{R}^3$ into camera t is therefore given by $\mathbf{u} = \pi(\mathbf{K}\mathbf{T}_k^t\bar{\mathbf{p}}_k)$ (where the de-homogenization is implied for notational clarity). We also define the inverse projection function $\mathbf{p} = \pi^{-1}(\mathbf{u}, d) = \bar{\mathbf{u}}/d$, which maps pixel $\mathbf{u}$ to 3D point $\mathbf{p}$ with depth $1/d$.

We define the kth keyframe to be a tuple $K_k = (\mathbf{S}_k^W, I_k, D_k, V_k)$, where $D_k : \Omega \to \mathbb{R}$ is the full-resolution inverse depthmap associated with image I_k (scaled to have a mean of 1), and V_k is the associated map of inverse depth variances. Note that only a subset of pixels $\Omega_k \subseteq \Omega$ will have valid inverse depth estimates. $\mathbf{S}_k^W = (\mathbf{T}_k^W, s_k) \in \text{Sim}(3)$ is the pose $\mathbf{T}_k^W$ of the camera with respect to world frame W (taken to be that of the first keyframe) and scale factor $s_k > 0$ which scales the geometry in D_k appropriately.

We arrange the keyframes in a pose graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, where $\mathcal{V} = \{K_k\}$ is the set of keyframes and $\mathcal{E} = \{\mathbf{S}_i^j \in \text{Sim}(3) : K_i, K_j \in \mathcal{V}\}$ is the set of constraint factors. Each $\mathbf{S}_i^j$ provides a measurement of the rigid body motion $\mathbf{T}_i^j$ and scale factor $s_i^j > 0$ that aligns the point clouds $\pi^{-1}(\Omega_i, D_i(\Omega_i))$ and $\pi^{-1}(\Omega_j, D_j(\Omega_j))$. The projection of all the keyframe point clouds into W will comprise our map.

B. Dense Monocular SLAM

Given a sequence of images I_t from a moving camera and the current designated keyframe K_k, our goal is to estimate online:

- The current camera pose $\mathbf{T}_t^k$ (Section IV-A)
- The inverse depthmap D_k (Sections IV-B to IV-E)
- The optimal keyframe pose $\mathbf{S}_k^W$ (Section IV-F).

While the above quantities are interdependent, we follow what has become the standard approach to monocular SLAM and decouple their computations, solving for each component in a separate thread in parallel.

Furthermore, we focus our attention on improving the quality of the depthmap D_k (which in turn affects the estimates $\mathbf{T}_t^k$ and $\mathbf{S}_k^W$). State-of-the-art approaches such as LSD-SLAM [14] only estimate depth for regions of Ω with high-image gradient, and are unable to interpolate through low-texture regions, resulting in point cloud maps with undesirable holes (that is $|\Omega_k| \ll |\Omega|$). Our primary contribution will be to increase the fraction of each keyframe with valid depth estimates (i.e. increase $|\Omega_k|$), while also increasing the accuracy of D_k, through a multi-resolution depth estimation process using quadtrees.

IV. Multi-Level Mapping

In this section we outline our dense monocular SLAM pipeline (see Figure 2). Section IV-A describes our $\mathbb{SE}(3)$ dense tracking front-end and Section IV-F summarizes our $\text{Sim}(3)$ pose-graph optimization backend, which wrap around the keyframe depth estimation component that we consider our primary contribution. In Section IV-B, we formulate our quadtree keyframe data structure and how we estimate depth at multiple image scales. Section IV-C outlines our hole-filling approach to further increase depthmap density, while Section IV-D describes the triangulation and rasterization procedure used to project the variable-resolution depthmaps back to the native image resolution for tracking and display. Section IV-E describes the final round of spatial regularization on the variable resolution depthmaps before they are incrementally aligned in the Sim(3) pose-graph and displayed as point clouds.

A. Tracking on $\mathbb{SE}(3)$

We use the coarse-to-fine image alignment method of [14] for tracking in $\mathbb{SE}(3)$ between keyframe K_k and the current image I_t, with the addition of a global illumination term to account for lighting variation between frames as in [22]. With the increased density of our keyframes, we also use all available pixels for tracking, not just those with high-image gradient at the finest image scale as in [14].

For each incoming frame I_t, we estimate $\mathbf{T}_t^k$ using weighted Gauss-Newton optimization with the following objective function:

$$E(\mathbf{T}_t^k) = \sum_{\mathbf{u}\in\Omega_k} \left|\left| \frac{r_p^2(\mathbf{u}, \mathbf{T}_t^k)}{\sigma^2_{r_p(\mathbf{p},\mathbf{T}_t^k)}} \right|\right|_\epsilon \tag{2}$$

where $||\cdot||_\epsilon$ is the Huber norm defined as

$$||x||_\epsilon = \begin{cases} \frac{||x||_2^2}{2\epsilon} & \text{if } ||x||_2 \le \epsilon \\ ||x||_1 - \frac{\epsilon}{2} & \text{otherwise.} \end{cases} \tag{3}$$

The photometric residual r_p is defined as

$$r_p(\mathbf{u}, \mathbf{T}_t^k) = I_k(\mathbf{u}) - I_t(\pi(\mathbf{K}\mathbf{p})) - r_{1/2} \quad (4)$$

$$\mathbf{p} = \mathbf{T}_k^t \mathbf{K}^{-1} \pi^{-1}(\mathbf{u}, D_k(\mathbf{u})) \quad (5)$$

$$\sigma^2_{r_p(\mathbf{p},\mathbf{T}_t^k)} = 2\sigma_I^2 + \left(\frac{\partial r_p(\mathbf{u}, \mathbf{T}_t^k)}{\partial D_k(\mathbf{u})}\right)^2 V_k(\mathbf{u}) \quad (6)$$

where $\mathbf{p}$ and $\sigma^2_{r_p(\mathbf{p},\mathbf{T}_t^k)}$ are the projected 3D point and variance of pixel $\mathbf{u}$ from the keyframe into the new frame assuming inverse depth $D_k(\mathbf{u})$. σ_I^2 is the user-set pixel intensity noise. The $r_{1/2}$ term is the median photometric residual across all pixels and serves to remove global illumination changes from the cost.

After convergence, we use the newly tracked frame $(I_t, \mathbf{T}_t^k)$ to update the depthmap D_k and variances V_k of K_k as described in the next sections.

B. Depth Estimation using Quadtree Keyframes

With $\mathbf{T}_t^k$ computed, (I_k, I_t) now form a stereo [1] pair that we use to update D_k. We follow the depth estimation approach in [13], but perform stereo computations at multiple resolutions to increase the density of the depthmap without sacrificing speed (see Figure 3).

For each stereo pair (I_k, I_t) we compute standard L-level power-of-two image pyramids that we denote $({}^L I_k, {}^L I_t)$. We interpret these pyramids as tree data structures, with each parent pixel (or *node*) connected to four child nodes at a finer image scale.

We then transform ${}^L I_k$ into a quadtree ${}^Q I_k$ by identifying sub-trees of ${}^L I_k$ with similar pixel intensities and *clipping* them from the tree [23]. We extract the *leaf* nodes of ${}^Q I_k$ and refer to them as ${}^q I_k$. [2] If we let $I_k^l : \Omega^l \to \mathbb{R}$ represent the lth image level in the full pyramid ${}^L I_k$, node $i \in {}^q I_k$ comprises a pixel location $\mathbf{u}_i \in \Omega^{l_i}$ with intensity $I_k^{l_i}(\mathbf{u}_i)$, where l_i is the pyramid level index. We then define a corresponding inverse depthmap ${}^q D_k$ with variances ${}^q V_k$ and perform updates on this representation before projecting the depths back to the full-resolution D_k (see Section IV-D).

To update the depth ${}^q D_k(i)$ for node $i \in {}^q I_k$, we check the magnitude of the image gradient $\nabla I_k^{l_i}(\mathbf{u}_i)$. If the magnitude falls below a threshold, we skip the update. If the gradient is sufficient, we search along the epipolar line defined by $\mathbf{T}_t^k$ in image $I_t^{l_i}$ for a matching pixel. [3] Matches are determined using sum-of-squared-differences (SSD) along a 5-sample window.

If a match is found, we compute a variance for this "measurement" according to the noise model in [13] and perform a Kalman update to ${}^q D_k(i)$ and ${}^q V_k(i)$. We keep track of the number of successful observations for each node and do not initiate the epipolar search if this number drops below a threshold (that is we mark the node as "invalid").

Our approach performs a similar number of stereo computations per keyframe as that of [13], but is able to more effectively "cover" the keyframe with inverse depth estimates by representing lower-texture image regions with coarser resolution pixels. We perform stereo computations at the image scale appropriate for the available texture, by which we are able to increase the density of the keyframe depthmap D_k after projecting the depths in ${}^q D_k$ back to full-resolution.

After the inverse depths for each node in the multi-level depthmap are updated, we attempt to fill the holes in the depthmap created by invalid nodes as described next section.

C. Hole-Filling

If the stereo computation at node $i \in {}^q I_k$ has failed repeatedly, we may still infer its depth by considering its spatial neighbors. If the number of successful stereo observations for the spatial neighbors of i exceeds a threshold, we initialize ${}^q D_k(i)$ to be the mean of the estimates of its neighbors, weighted by the variance of each estimate, and attempt the stereo search again at the next incoming frame. This hole-filling procedure helps to increase the density of ${}^q D_k$ and ensure that as many pixels as possible have depth estimates. After a round of hole-filling, we project ${}^q D_k$ back to the full-resolution D_k, as described in Section IV-D, which is then passed to the tracking frontend described in Section IV-A.

D. Triangulation and Rasterization

We perform depthmap updates on the variable resolution representation ${}^q D_k$ to enable spatial regularization and hole filling, but need to track the next incoming image and display point clouds using the best full-resolution keyframe depthmap we can infer so far. Thus, at each timestep, we take the current variable resolution keyframe depthmap ${}^q D_k$ and recover the native resolution depthmap D_k.

For each pixel $\mathbf{u} \in \Omega$ at native resolution, we approximate $D_k(\mathbf{u})$ by linearly interpolating among the depth estimates at nearby quadtree leaves in ${}^q D_k$ using a simple triangulation and rasterization scheme. This kind of interpolation is not strictly necessary, but assigning the same depth to all full-resolution pixels corresponding to a leaf node in ${}^q D_k$ leads to unnecessarily quantized or "blocky" depthmaps (this is effectively a piecewise constant approximation versus a piecewise linear approximation). The interpolation scheme then raises the question of which quadtree leaves are neighbors to a particular full-resolution pixel $\mathbf{u}$, and we use a simple triangulation scheme to determine the neighbors [24].

Given a triangulation of ${}^q D_k$, we linearly interpolate between the multi-level depth estimates using software rasterization accelerated by SIMD instructions to fill in D_k. [4] We compute a new interpolated depthmap for every mapping iteration to pass to the $\mathbb{SE}(3)$ tracker. However, we use a variational regularizer described in the next section to remove

[1] Stereo in this case is across pairs of successive monocular images, rather than simultaneous images from binocular cameras. We follow [13], [14], and others in using the term "stereo" for this monocular image processing.

[2] An equivalent construction of ${}^q I_k$ would be to take I_k and recursively merge pixel neighborhoods with similar intensities into single pixels defined at coarser image scales.

[3] Note that the comparison is performed between the quadtree leaf nodes of the keyframe image and the full image pyramid for the incoming frame. We do not need to compute a quadtree representation for the incoming frame.

[4] Note that we linearly interpolate the *depths* associated with each node, not the inverse depths.

outliers and smooth away noise before Sim(3) alignment and point cloud display.

E. Spatial Regularization

The multi-resolution inverse depthmap ${}^{q}D_k$ computed in Section IV-B may be corrupted by noise as well as outliers from false-matches that can degrade map quality. Before we finalize a keyframe and pass its depthmap to the pose-graph optimization backend, therefore, we use the first-order primal-dual optimization algorithm of [15] to remove the outliers and smooth away noise.

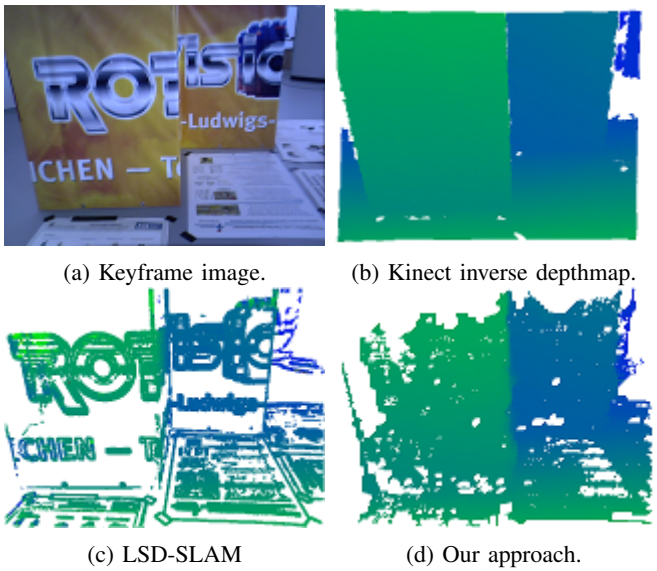
While this approach is typically implemented on a GPU [8], [10], [11], we find that running in a separate thread and operating on our quadtree-compressed depthmap, our version runs sufficiently fast for real-time operation on a CPU. After a new keyframe is triggered (based on the euclidean and angular distance to the last keyframe), we run our regularizer for a fixed number of iterations on the outgoing depthmap before passing it to the tracker and pose-graph optimizer. [5]

Assuming ${}^{q}I_k$ contains N nodes, we first arrange ${}^{q}D_k$ into a vector $\mathbf{z} = \begin{bmatrix} {}^{q}D_k(1) & \dots & {}^{q}D_k(N) \end{bmatrix}^T$. We let the vector $\boldsymbol{\xi} \in \mathbb{R}^N$ denote the regularized solution and minimize the following convex objective function:

$$E(\boldsymbol{\xi}) = TV_\epsilon(\boldsymbol{\xi}) + \lambda ||\mathbf{W}(\boldsymbol{\xi} - \mathbf{z})||_1 \tag{7}$$

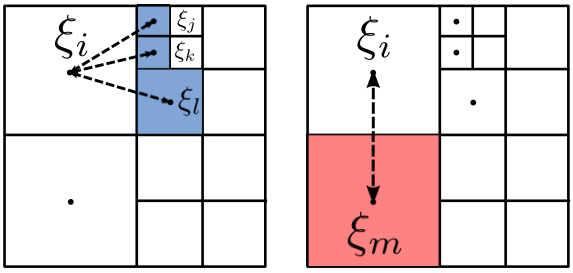
where $TV_\epsilon(\boldsymbol{\xi})$ is the Total Variation-Huber norm, λ is a scale-factor that sets the influence of the L_1 data-term, and $\mathbf{W}$ is a diagonal weighting matrix. The Total Variation-Huber norm promotes smooth solutions while preserving edges and the weighted L_1 data-term reduces the effect of outliers in $\mathbf{z}$.

[5] We find that running the regularizer after the depth estimation process is complete produces better results than applying the regularization in parallel, before the depths have converged.



(a) Keyframe image. (b) Kinect inverse depthmap. (c) LSD-SLAM (d) Our approach.

Fig. 4: Our algorithm significantly increases the number of accurate inverse depth estimates per keyframe compared to LSD-SLAM [14], a state-of-the-art algorithm for semi-dense monocular SLAM.



(a) Horizontal Neighbors (b) Vertical Neighbors

Fig. 5: *Discrete derivative computation*: We approximate the discrete derivative at node ξ_i in our multi-level inverse depthmap using forward-differences with the node's (a) horizontal and (b) vertical neighbors.

In the discrete setting, the Total Variation-Huber norm is defined as

$$TV_\epsilon(\boldsymbol{\xi}) = ||\mathbf{D}\boldsymbol{\xi}||_\epsilon \tag{8}$$

for an appropriate discrete gradient operator $\mathbf{D} : \mathbb{R}^N \rightarrow \mathbb{R}^{2N}$ that captures the horizontal and vertical derivatives for the multi-scale $\boldsymbol{\xi}$.

If $\boldsymbol{\xi}$ was defined on a regular grid at a single scale, $\mathbf{D}$ could simply be the sparse matrix that implements the standard horizontal and vertical forward differences:

$$(\nabla\boldsymbol{\xi})^h_{i,j} = \boldsymbol{\xi}_{i+1,j} - \boldsymbol{\xi}_{i,j} \tag{9}$$

$$(\nabla\boldsymbol{\xi})^v_{i,j} = \boldsymbol{\xi}_{i,j+1} - \boldsymbol{\xi}_{i,j}. \tag{10}$$

In our case, we must modify $\mathbf{D}$ such that the horizontal and vertical derivatives are approximated on our variable-scale $\boldsymbol{\xi}$. As shown in Figure 5, each element of $\boldsymbol{\xi}$ corresponds to a square region of pixels at full-resolution. We approximate the derivative in each direction by simply averaging the forward differences between a node and the nodes bordering its square region:

$$(\nabla\boldsymbol{\xi})^h_i = \left(\frac{1}{|\mathcal{N}^h(\boldsymbol{\xi}_i)|} \sum_{j=1}^{|\mathcal{N}^h(\boldsymbol{\xi}_i)|} \mathcal{N}^h_j(\boldsymbol{\xi}_i) \right) - \boldsymbol{\xi}_i \tag{11}$$

$$(\nabla\boldsymbol{\xi})^v_i = \left(\frac{1}{|\mathcal{N}^v(\boldsymbol{\xi}_i)|} \sum_{j=1}^{|\mathcal{N}^v(\boldsymbol{\xi}_i)|} \mathcal{N}^v_j(\boldsymbol{\xi}_i) \right) - \boldsymbol{\xi}_i, \tag{12}$$

where $\mathcal{N}^h(\boldsymbol{\xi}_i)$ and $\mathcal{N}^v(\boldsymbol{\xi}_i)$ denote the horizontal and vertical neighbors of $\boldsymbol{\xi}_i$, respectively.

The diagonal weighting matrix $\mathbf{W} = \text{diag}\,(w_1, \dots, w_N)$ incorporates the depth uncertainty into the data term and is defined as

$$w_i = \begin{cases} 0 & \text{if no stereo match found} \\ \frac{1}{\sqrt{{}^{q}V_k(i)}} & \text{otherwise.} \end{cases} \tag{13}$$

To derive the update equations to minimize (7), we first compute F^* (the convex conjugate of $TV_\epsilon(\boldsymbol{\xi})$) using the dual variable $\mathbf{q} = \begin{bmatrix} q_1^T & \dots & q_N^T \end{bmatrix}^T \in \mathbb{R}^{2N}$:

$$F^*(\mathbf{q}) = \frac{\epsilon}{2}\mathbf{q}^T\mathbf{q} - \delta_Q(\mathbf{q}), \tag{14}$$

(a) Desk dataset.

(b) Bench dataset.

Fig. 6: We demonstrate the quality of our reconstructions on several datasets recorded using a handheld camera: (a) shows the final point cloud map from a small desk scene, while (b) shows the map from an outdoor bench area. All processing was performed in real-time on a consumer laptop.

where we utilize the indicator function

$$\delta_Q(x) = \begin{cases} 0 & \text{if } x \in Q \\ \infty & \text{otherwise} \end{cases} \tag{15}$$

for the set $Q = \{\mathbf{q} \in \mathbb{R}^{2N} : ||q_i||_2 \leq 1 \text{ for } i = 1, \ldots, N\}$.

Letting $G(\boldsymbol{\xi}) = ||\mathbf{W}(\boldsymbol{\xi} - \mathbf{z})||_1$, the primal-dual update steps following [15] are given by:

$$\mathbf{q}^{n+1} = \text{prox}_{\alpha_\mathbf{q}, F^*}(\mathbf{q}^n + \alpha_\mathbf{q} \mathbf{D} \bar{\boldsymbol{\xi}}) \tag{16}$$

$$\boldsymbol{\xi}^{n+1} = \text{prox}_{\alpha_{\boldsymbol{\xi}}, G}(\boldsymbol{\xi}^n - \alpha_{\boldsymbol{\xi}} \mathbf{D}^T \mathbf{q}^{n+1}) \tag{17}$$

$$\bar{\boldsymbol{\xi}}^{n+1} = \boldsymbol{\xi}^{n+1} + \theta(\boldsymbol{\xi}^{n+1} - \boldsymbol{\xi}^n) \tag{18}$$

for step sizes $\alpha_{\boldsymbol{\xi}}, \alpha_\mathbf{q} > 0$, $\theta \in [0, 1]$. The proximal operators generalize gradient steps to non-differentiable functions and are given point-wise by

$$\text{prox}_{\alpha_\mathbf{q}, F^*}(y_i) = \frac{\frac{y_i}{1+\alpha_\mathbf{q}\epsilon}}{\max\{1, ||\frac{y_i}{1+\alpha_\mathbf{q}\epsilon}||_2\}} \tag{19}$$

$$\text{prox}_{\alpha_{\boldsymbol{\xi}}, G}(x_i) = \begin{cases} x_i - \lambda w_i \alpha_{\boldsymbol{\xi}} & \text{if } x_i - z_i > \lambda \alpha_{\boldsymbol{\xi}} w_i \\ x_i + \lambda w_i \alpha_{\boldsymbol{\xi}} & \text{if } x_i - z_i < -\lambda \alpha_{\boldsymbol{\xi}} w_i \\ z_i & \text{if } |x_i - z_i| \leq \lambda \alpha_{\boldsymbol{\xi}} w_i. \end{cases} \tag{20}$$

After convergence, we copy $\boldsymbol{\xi}$ back to ${}^q D_k$ and rasterize the solution to obtain the smoothed depthmap D_k, which is then incrementally aligned with overlapping keyframes

Fig. 7: We validate our monocular SLAM pipeline on challenging benchmark datasets captured using a Microsoft Kinect [25]. The `fr3/structure_texture_far` dataset shown above demonstrates our algorithm's ability to produce dense reconstructions through low-texture image regions.

in our pose-graph optimization backend and displayed (see Section IV-F).

F. Pose-graph Optimization on $Sim(3)$

Similar to our $\mathbb{SE}(3)$ tracking front-end, we use the robust image and depth alignment method of [14] for generating constraints in our $\text{Sim}(3)$ pose-graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ in order to estimate the optimal keyframe poses $\mathbf{S}_k^W$ and point cloud.

When keyframe K_k is finalized, it is added to $\mathcal{V}$ and a set of potential neighbor keyframes $\mathcal{C} \subseteq \mathcal{V} \setminus K_k$ is generated through a search of predecessors in $\mathcal{V}$ and an appearance-based loop-closure detector [26].

For each $K_j \in \mathcal{C}$, transforms $\mathbf{S}_k^j, \mathbf{S}_j^k \in \text{Sim}(3)$ linking K_k and K_j are computed using the $\text{Sim}(3)$ tracking method described below. If $\mathbf{S}_k^j$ and $\mathbf{S}_j^k$ are consistent with each other, they are added to the constraint set $\mathcal{E}$. $\mathcal{G}$ is then refined online using an open-source optimization package [27] to produce the optimal keyframe pose $\mathbf{S}_k^W$, which we use to project D_k into the world frame W.

Tracking on $\text{Sim}(3)$ is achieved using weighted Gauss-Newton optimization as in Section IV-A, with the cost function from 2 modified to include the residual r_d between the two depthmaps:

$$E(\mathbf{S}_k^j) = \sum_{\mathbf{u} \in \Omega_k} \left\| \frac{r_p^2(\mathbf{u}, \mathbf{S}_k^j)}{\sigma^2_{r_p(\mathbf{p}, \mathbf{S}_k^j)}} + \frac{r_d^2(\mathbf{u}, \mathbf{S}_k^j)}{\sigma^2_{r_d(\mathbf{p}, \mathbf{S}_k^j)}} \right\|_\epsilon \tag{21}$$

$$r_d(\mathbf{u}, \mathbf{S}_k^j) = [\mathbf{p}]_z^{-1} - D_j(\pi(\mathbf{K}\mathbf{p})) \tag{22}$$

$$\mathbf{p} = \mathbf{S}_k^j \mathbf{K}^{-1} \pi^{-1}(\mathbf{u}, D_k(\mathbf{u})) \tag{23}$$

$$\sigma^2_{r_d(\mathbf{u}, \mathbf{S}_k^j)} = J_k^2 V_k(\mathbf{u}) + J_j^2 V_j(\pi(\mathbf{K}\mathbf{p})) \tag{24}$$

$$J_k = \frac{\partial r_d(\mathbf{u}, \mathbf{S}_k^j)}{\partial D_k(\mathbf{u})}, \quad J_j = \frac{\partial r_d(\mathbf{u}, \mathbf{S}_k^j)}{\partial D_j(\pi(\mathbf{p}))}. \tag{25}$$

V. Evaluation

We evaluated our tracking and reconstructions qualitatively using video captured from a handheld camera as well

Relative Inverse Depth Error [%]

Dataset	[14]	Ours
fr2/desk	19	**17**
fr3/long_office_household	31	**20**
fr3/nostructure_texture_near_withloop	8.5	**6.2**
fr3/structure_texture_far	4.4	**2.7**

TABLE I: For each keyframe, we compute the inverse depth error relative to the corresponding value from the Kinect across all pixels with valid estimates. The results from each dataset are averaged across all keyframes and across 10 trials. Our multi-level approach achieves more accurate depth estimates on all four datasets.

Average Keyframe Density [%]

Dataset	[14]	Ours
fr2/desk	18	**26**
fr3/long_office_household	16	**23**
fr3/nostructure_texture_near_withloop	20	**41**
fr3/structure_texture_far	37	**63**

TABLE II: We compute the fraction of pixels in each keyframe with inverse depth estimates that are within 10 percent of the corresponding values from the Kinect depth frames. The results for each dataset are averaged across all keyframes and across 10 trials. Our multi-level approach significantly increases the number of accurate inverse depths per keyframe.

Tracking Accuracy [RMSE]

	Pos. [m]		Angle [deg]	
Dataset	[14]	Ours	[14]	Ours
fr2/desk	2.1	**0.15**	88	**4.8**
fr3/long_office_household	2.1	**0.65**	68	**11**
fr3/nostructure_texture_near_withloop	0.28	**0.22**	3.7	**2.6**
fr3/structure_texture_far	0.22	**0.17**	2.3	**0.83**

TABLE III: The addition of a robust illumination term in our $\mathbb{SE}(3)$ tracker, coupled with the increased depthmap density, results in improved tracking performance. LSD-SLAM had particular trouble with `fr2/desk` and `fr3/long_office_household` and would frequently lose track of the camera or fail to detect a loop closure, resulting in increased error. The results for each dataset are computed across 10 trials.

Average Time per Mapping Update [ms]

Dataset	[14]	Ours
fr2/desk	16	17
fr3/long_office_household	13	16
fr3/nostructure_texture_near_withloop	12	15
fr3/structure_texture_far	14	17

TABLE IV: Our system provides significantly more depth estimates per keyframe, while maintaining real-time operation at 30 Hz. Here we present the average run-time per keyframe update, including depth estimation, hole-filling, and rasterization for each dataset over 10 trials.

quantitatively on publicly available benchmark datasets [25]. Our implementation was based off the LSD-SLAM source code [6]. All processing was done in real time on a consumer laptop with an Intel Core i7 processor with 8 GB of RAM. All metrics were computed using the raw inverse depthmaps, however, we set a maximum depth threshold for display purposes.

A. Qualitative Evaluation

We captured several video sequences using a global shutter Point Grey Firefly color camera with a resolution of 640x480 pixels and 30 Hz frame-rate. The first small-scale sequence was recorded in an office area and is shown in Figure 6a. Note the accurate reconstructions of high-texture regions around the desk. The second sequence was recorded outside near a set of benches shown in Figure 6b. The final sequence was captured inside a laboratory test space and is shown in Figure 1. These examples show how our multi-level approach is able to interpolate through low-texture regions such as the ground and walls.

B. Quantitative Evaluation

We ran our pipeline on four video sequences from the TUM RGB-D SLAM Benchmarks, which were captured using a hand-held Microsoft Kinect in a variety of environments [25] with pose ground truth provided by a motion capture system. We used the depth frames as a proxy for depth ground truth and compare the inverse depth accuracy (Table I), keyframe density (Table II), tracking accuracy (Table III), and mapping update time (Table IV) against LSD-SLAM [14]. We use the first depth frame to initialize our system in order to set the global scale factor and ignore the first 5 keyframes to screen out initialization effects. Furthermore, as both our pipeline and LSD-SLAM are heavily multi-threaded, performance can vary from run to run based on differences in keyframe selection and loop closure detection, so we report metrics averaged over 10 trials for each dataset.

We note that these benchmark videos are challenging for pure monocular SLAM systems due to the automatic settings (gain, exposure, brightness, etc.) and rolling shutter of the Kinect's RGB camera as well as camera motion. LSD-SLAM had particular trouble with the first two datasets (`fr2/desk` and `fr3/long_office_household`) and would frequently lose track of the camera or fail to detect a loop closure, leading to increased tracking error over the 10 trials. Our system shows significant improvement over LSD-SLAM, with a substantial increase in the fraction of each keyframe with accurate depth estimates, as well as more accurate tracking, while maintaining real-time operation (See Figure 4 and Figure 7).

VI. Conclusion

We have presented a monocular SLAM algorithm capable of operating in real-time without GPU acceleration that significantly improves upon the state-of-the-art in terms of

[6] `https://github.com/tum-vision/lsd_slam`

tracking accuracy, reconstruction accuracy, and reconstruction density. Our method extends the semi-dense solution of [14] to produce dense point clouds while being able to run online at 30 Hz. Our depth estimation component intelligently distributes resources to image regions with high-texture while estimating low-texture regions at coarser resolutions, achieving a substantial speedup. In addition, this allows for low-frequency information to be exploited in the depth estimation. The noisy, multi-level inverse depthmaps are then smoothed using a variational regularizer before being triangulated and rasterized back up to the native image scale and added to a keyframe pose-graph. The increased keyframe density also improves tracking performance when combined with a global illumination term to account for lighting differences between frames.

REFERENCES

[1] D. G. Lowe, "Object recognition from local scale-invariant features," in *Proc. ICCV*, 1999.

[2] H. Bay, A. Ess, T. Tuytelaars, and L. Van Gool, "Speeded-up robust features (SURF)," *Comp. Vis. and Image Understanding*, 2008.

[3] E. Rublee, V. Rabaud, K. Konolige, and G. Bradski, "ORB: An efficient alternative to SIFT or SURF," in *Proc. ICCV*, 2011.

[4] D. Scaramuzza and F. Fraundorfer, "Visual Odometry: Part 1: The First 30 Years and Fundamentals," *IEEE Robotics & Automation Magazine*, 2011.

[5] F. Fraundorfer and D. Scaramuzza, "Visual Odometry: Part 2: Matching, Robustness, Optimization, and Applications," *IEEE Robotics & Automation Magazine*, 2012.

[6] A. J. Davison, "Real-time simultaneous localisation and mapping with a single camera," in *Proc. ICCV*, 2003.

[7] G. Klein and D. Murray, "Parallel tracking and mapping for small AR workspaces," in *Proc. ISMAR*, 2007.

[8] R. A. Newcombe, S. J. Lovegrove, and A. J. Davison, "DTAM: Dense tracking and mapping in real-time," in *Proc. ICCV*, 2011.

[9] V. Pradeep, C. Rhemann, S. Izadi, C. Zach, M. Bleyer, and S. Bathiche, "MonoFusion: Real-time 3D reconstruction of small scenes with a single web camera," in *Proc. ISMAR*, 2013.

[10] M. Pizzoli, C. Forster, and D. Scaramuzza, "REMODE: Probabilistic, monocular dense reconstruction in real time," in *Proc. ICRA*, 2014.

[11] G. Graber, T. Pock, and H. Bischof, "Online 3D reconstruction using convex optimization," in *Proc. ICCV workshop*, 2011.

[12] C. Forster, M. Pizzoli, and D. Scaramuzza, "SVO: Fast semi-direct monocular visual odometry," in *Proc. ICRA*, 2014.

[13] J. Engel, J. Sturm, and D. Cremers, "Semi-dense visual odometry for a monocular camera," in *Proc. ICCV*, 2013.

[14] J. Engel, T. Schöps, and D. Cremers, "LSD-SLAM: large-scale direct monocular slam," *Proc. ECCV*, 2014.

[15] A. Chambolle and T. Pock, "A first-order primal-dual algorithm for convex problems with applications to imaging," *Journal of Mathematical Imaging and Vision*, 2011.

[16] R. Yang and M. Pollefeys, "Multi-resolution real-time stereo on commodity graphics hardware," in *Proc. CVPR*, 2003.

[17] M. Gong and Y.-H. Yang, "Multi-resolution stereo matching using genetic algorithm," in *Proc. SMBV*, 2001.

[18] D. Gallup, J.-M. Frahm, P. Mordohai, and M. Pollefeys, "Variable baseline/resolution stereo," in *Proc. CVPR*, 2008.

[19] J. Stückler and S. Behnke, "Integrating depth and color cues for dense multi-resolution scene mapping using RGB-D cameras," in *Proc. MFI*, 2012.

[20] A. Martinelli, A. Tapus, K. O. Arras, and R. Siegwart, "Multi-resolution SLAM for real world navigation," in *Proc. ISRR*, 2005.

[21] D. Chekhlov, M. Pupilli, W. Mayol-Cuevas, and A. Calway, "Real-time and robust monocular SLAM using predictive multi-resolution descriptors," in *Advances in Visual Computing*, 2006.

[22] T. Gonçalves, A. Comport, *et al.*, "Real-time direct tracking of color images in the presence of illumination variation," in *Proc. ICRA*, 2011.

[23] R. A. Finkel and J. L. Bentley, "Quad Trees: A data structure for retrieval on composite keys," *Acta Informatica*, 1974.

[24] M. De Berg, M. Van Kreveld, M. Overmars, and O. C. Schwarzkopf, *Computational Geometry*. Springer, 2000.

[25] J. Sturm, N. Engelhard, F. Endres, W. Burgard, and D. Cremers, "A benchmark for the evaluation of RGB-D slam systems," in *Proc. IROS*, 2012.

[26] A. Glover, W. Maddern, M. Warren, S. Reid, M. Milford, and G. Wyeth, "OpenFABMAP: An open source toolbox for appearance-based loop closure detection," in *Proc. ICRA*, 2012.

[27] R. Kümmerle, G. Grisetti, H. Strasdat, K. Konolige, and W. Burgard, "g2o: a general framework for graph optimization," in *Proc. ICRA*, 2011.