FLaME: Fast Lightweight Mesh Estimation using Variational Smoothing on Delaunay Graphs

W. Nicholas Greene   Nicholas Roy
Computer Science and Artificial Intelligence Laboratory
Massachusetts Institute of Technology
{wng,nickroy}@csail.mit.edu

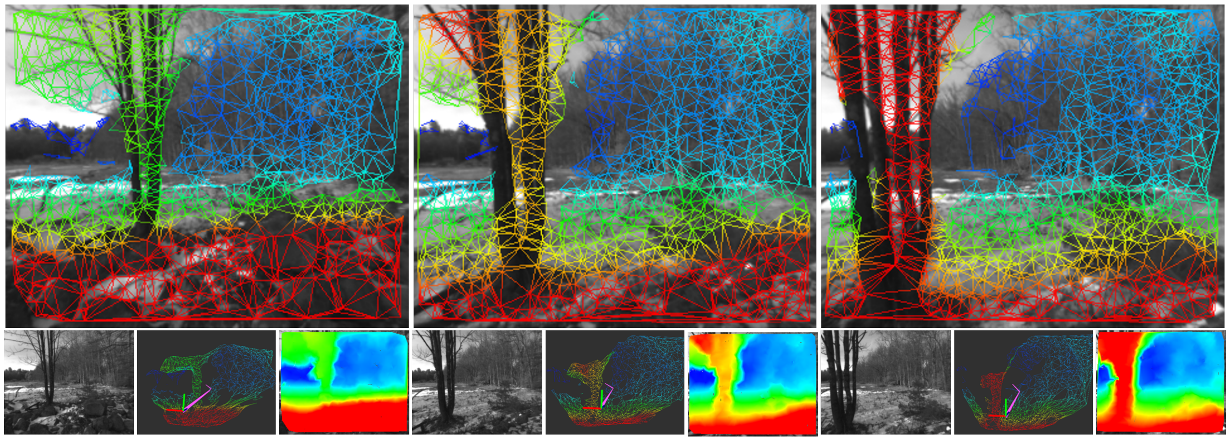

Figure 1: *Fast Lightweight Mesh Estimation:* FLaME generates 3D mesh reconstructions from monocular images in real-time onboard computationally constrained platforms. The key to the approach is a graph-based variational optimization framework that allows for the mesh to be efficiently smoothed and refined. The top row of images (from left to right) show the meshes computed onboard a small autonomous robot flying at 3.5 meters-per-second as it avoids a tree. The bottom row shows the current frame (left), the collision-free plan in pink (middle), and the dense depthmap generated from the mesh (right) for each timestep along the approach.

Abstract

We propose a lightweight method for dense online monocular depth estimation capable of reconstructing 3D meshes on computationally constrained platforms. Our main contribution is to pose the reconstruction problem as a non-local variational optimization over a time-varying Delaunay graph of the scene geometry, which allows for an efficient, keyframeless approach to depth estimation. The graph can be tuned to favor reconstruction quality or speed and is continuously smoothed and augmented as the camera explores the scene. Unlike keyframe-based approaches, the optimized surface is always available at the current pose, which is necessary for low-latency obstacle avoidance.

FLaME (Fast Lightweight Mesh Estimation) can generate mesh reconstructions at upwards of 230 Hz using less than one Intel i7 CPU core, which enables operation on size, weight, and power-constrained platforms. We present results from both benchmark datasets and experiments running FLaME in-the-loop onboard a small flying quadrotor.

1. Introduction

Estimating dense 3D geometry from 2D images taken from a moving monocular camera is a fundamental problem in computer vision with a wide range of applications in robotics and augmented reality (AR). Though the visual tracking component of monocular *simultaneous localization and mapping* (SLAM) has reached a certain level of maturity over the last ten years [13, 9, 12, 10, 8, 14, 15], efficiently reconstructing dense environment representations for autonomous navigation or AR on small size-weight-and-power (SWaP) constrained platforms (such as mobile robots and smartphones) is still an active research front. Current approaches either transmit information to a groundstation for processing [28, 1, 5], sacrifice density [7, 6, 10, 14], run at significantly reduced framerates [21, 22], or limit the reconstruction volume to small scenes [16] or to past keyframes [11], all of which restrict their utility in practice, especially for mobile robot navigation. In this paper, we propose a novel monocular depth estimation pipeline that

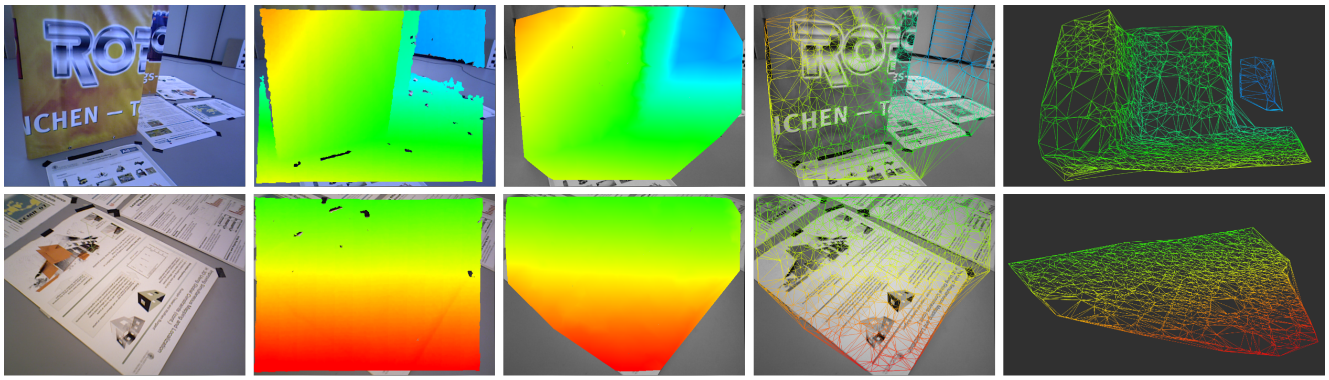
Figure 2: *Second Order Smoothing:* FLaME estimates dense inverse depth meshes by optimizing a non-local, second-order variational smoothness cost over a semi-sparsely sampled *Delaunay Graph*. Minimizing this cost function promotes piecewise planar structure as shown above on benchmark data collected from a handheld Kinect [27]. From left to right, each column shows the input RGB image, the Kinect depthmap, the generated FLaME depthmap, the FLaME mesh in the current view, and the FLaME mesh projected into 3D. Note the smooth planar reconstructions that are enabled by the approach and the accuracy of the depthmaps relative to those from the Kinect.

enables dense geometry to be efficiently computed at upwards of 230 Hz using less than one Intel i7 CPU core — a small enough footprint to fit completely onboard an autonomous micro-aerial vehicle (MAV), with sufficient accuracy to enable closed-loop motion planning using the reconstructions.

Our key insight is to recognize that for many environments and applications, the "every-pixel" methods that are currently in vogue for dense depth estimation massively oversample scenes relative to **(a)** their true geometric complexity, **(b)** the observable geometric complexity given the available texture and baseline, and **(c)** the additional computational effort required to spatially regularize (i.e. smooth) noisy and outlier-prone depth estimates. Oversampling the scene requires more computation for stereo matching, an unnecessary cost for pixels whose depth might be either redundant given the geometry or weakly observable given the environment texture or camera motion, and significantly increases the sophistication and runtime of regularization needed to produce plausible reconstructions (which often requires what amounts to batch optimizations over select *keyframes* in the past while the camera continues exploring).

Instead of a dense every-pixel approach, we propose a novel alternative that we call FLaME (*Fast Lightweight Mesh Estimation*) that directly estimates a *triangular mesh* of the scene (similar to the stereo work of [18]) and is advantageous for several reasons. First, meshes are more compact, efficient representations of the geometry and therefore require fewer depth estimates to encode the scene for a given level of detail. Second, by interpreting the mesh as a graph we show that we can exploit its connectivity structure to apply (and accelerate) state-of-the-art second-order variational regularization techniques that otherwise require GPUs to run online. Third, by reformulating the regularization objective in terms of the vertices and edges of this graph, we allow the smoothing optimization to be both *incremental* (in that new terms can be trivially added and removed as the graph is modified over time) and *keyframeless* (in that the solution can be easily propagated in time without restarting the optimization).

We show significant improvements over existing approaches on benchmark data in terms of runtime, CPU load, density, and accuracy, and present results from flight experiments running FLaME in-the-loop onboard a small MAV flying at up to 3.5 meters-per-second (see Figure 1 and 2).

2. Preliminaries

We first detail the notation used in the rest of the paper (Section 2.1) and give a brief overview of variational smoothing methods (Section 2.2) before describing the approach in detail (Section 3).

2.1. Notation

We represent images as scalar functions defined over the pixel domain $\Omega \subset \mathbb{R}^2$, such that $I_k : \Omega \to \mathbb{R}$ denotes the image taken at time index k. We represent the pose of the camera at time k with respect to the world frame W by the transform $\mathbf{T}_k^W \in \mathbb{SE}(3)$.

We let $\mathbf{K} \in \mathbb{R}^{3\times3}$ denote the intrinsic camera matrix. Vectors in homogeneous coordinates are given by $\bar{\mathbf{x}} = \begin{bmatrix}\mathbf{x}^T & 1\end{bmatrix}^T$. The functions $\pi(x, y, z) = (x/z, y/z)$ and $\pi^{-1}(\mathbf{u}, d) = \mathbf{K}^{-1}(d \cdot \bar{\mathbf{u}})$ denote the perspective projection function and its inverse for pixel $\mathbf{u} \in \Omega$ given depth d, respectively. The projection of a point $\mathbf{p}_W \in \mathbb{R}^3$ in the world into the camera at time k is therefore $\mathbf{u} = \pi(\mathbf{K}\mathbf{T}_W^k \bar{\mathbf{p}}_W)$ (the de-homogenization is implied). Finally, the *inverse* depthmap at time k is given by the function $\xi_k : \Omega \to \mathbb{R}_+$.

2.2. Variational Smoothing

In the continuous setting, variational methods seek to minimize objective functionals of the following form:

$$E(f) = E_{smooth}(f) + \lambda E_{data}(f), \tag{1}$$

for scalar function $f : \mathcal{X} \rightarrow \mathbb{R}$ and various choices of smoothness term E_{smooth}, data fidelity term E_{data}, and scalar $\lambda > 0$ that controls the balance of data-fitting versus smoothness.

When a noisy signal $z : \mathcal{X} \rightarrow \mathbb{R}$ is observed, a common choice of $E_{data}(f)$ is an outlier-robust L_1 norm:

$$E_{data}(f) = \int_{\mathcal{X}} |f(\mathbf{x}) - z(\mathbf{x})| \, d\mathbf{x}. \tag{2}$$

One powerful choice of E_{smooth} is the second-order Total Generalized Variation (TGV^2) semi-norm of [2]:

$$\begin{aligned} \text{TGV}^2(f) = \min_{\mathbf{w}(\mathbf{x}) \in \mathbb{R}^2} \alpha \int_{\mathcal{X}} |\nabla f(\mathbf{x}) - \mathbf{w}(\mathbf{x})| \; d\mathbf{x} \quad + \\ \beta \int_{\mathcal{X}} |\nabla \mathbf{w}(\mathbf{x})| \; d\mathbf{x}, \end{aligned} \tag{3}$$

which introduces auxiliary function $\mathbf{w} : \mathcal{X} \rightarrow \mathbb{R}^2$ and weights $\alpha, \beta \geq 0$. This functional penalizes discontinuities in the first two derivatives of f and promotes piecewise affine solutions. The contributions of the first and second derivatives to the overall cost are controlled by α and β.

It is important to note that this functional only incorporates *local* information through the gradient operator. A non-local extension (NLTGV^2) was therefore developed in [20] so that information beyond immediately neighboring pixels could influence the objective:

$$\begin{aligned} &\text{NLTGV}^2(f) = \\ &\min_{\mathbf{w}(\mathbf{x}) \in \mathbb{R}^2} \int_{\mathcal{X}} \int_{\mathcal{X}} \alpha(\mathbf{x}, \mathbf{y}) \, |f(\mathbf{x}) - f(\mathbf{y}) - \\ &\qquad \langle \mathbf{w}(\mathbf{x}), \mathbf{x} - \mathbf{y} \rangle| \, d\mathbf{x} d\mathbf{y} + \\ &\int_{\mathcal{X}} \int_{\mathcal{X}} \beta(\mathbf{x}, \mathbf{y}) \, |w_1(\mathbf{x}) - w_1(\mathbf{y})| \, d\mathbf{x} d\mathbf{y} + \\ &\int_{\mathcal{X}} \int_{\mathcal{X}} \beta(\mathbf{x}, \mathbf{y}) \, |w_2(\mathbf{x}) - w_2(\mathbf{y})| \, d\mathbf{x} d\mathbf{y}, \end{aligned} \tag{4}$$

for $\mathbf{w}(\mathbf{x}) = (w_1(\mathbf{x}), w_2(\mathbf{x}))$ and weight functions $\alpha(\mathbf{x}, \mathbf{y}) \geq 0$ and $\beta(\mathbf{x}, \mathbf{y}) \geq 0$, which encode the weighted, non-local gradients.

The work of Pinies et al. showed that when f is interpreted as an inverse depthmap ξ, smoothing with NLTGV^2 leads not only to piecewise affine solutions over $\mathcal{X} = \Omega$, but also over $\mathbb{R}^3$ when ξ is projected into 3D using π^{-1} (a non-trivial result) [19].

Although the choices of E_{data} and E_{smooth} outlined above are not differentiable, they are convex and can thus be efficiently minimized using convex optimization techniques. One popular optimization scheme is the first-order, primal-dual method of Chambolle and Pock [4], which solves optimization problems of the following form:

Algorithm 1 Method of Chambolle and Pock [4]

```
// Choose $\sigma, \tau > 0, \theta \in [0, 1]$.
while not converged do
    $\mathbf{q}^{k+1} = \text{prox}_{F^*}(\mathbf{q}^k + \sigma \mathbf{D} \bar{\mathbf{x}}^k)$
    $\mathbf{x}^{k+1} = \text{prox}_G(\mathbf{x}^k - \tau \mathbf{D}^* \mathbf{q}^{k+1})$
    $\bar{\mathbf{x}}^{k+1} = \mathbf{x}^{k+1} + \theta(\mathbf{x}^{k+1} - \mathbf{x}^k)$
```

$$\min_{\mathbf{x} \in \mathbb{R}^n} F(\mathbf{D}\mathbf{x}) + G(\mathbf{x}), \tag{5}$$

where $F : \mathbb{R}^m \rightarrow \mathbb{R}_+$ and $G : \mathbb{R}^n \rightarrow \mathbb{R}_+$ are convex and $\mathbf{D} : \mathbb{R}^n \rightarrow \mathbb{R}^m$ is a linear operator that usually encodes discrete gradients. The essence of the Chambolle and Pock approach is to represent F in terms of its *convex conjugate* and dual variable $\mathbf{q} \in \mathbb{R}^m$, resulting in the following saddlepoint problem:

$$\min_{\mathbf{x} \in \mathbb{R}^n} \max_{\mathbf{q} \in \mathbb{R}^m} \langle \mathbf{D}\mathbf{x}, \mathbf{q} \rangle - F^*(\mathbf{q}) + G(\mathbf{x}). \tag{6}$$

Optimal values for primal variable $\mathbf{x}$ and dual variable $\mathbf{q}$ are then obtained by repeated application of the *proximal* operator that generalizes gradient descent to non-differentiable functions [17] (see Algorithm 1).

3. Algorithm Overview

FLaME directly estimates an *inverse depth mesh* of the environment that efficiently encodes the scene geometry and allows for efficient, incremental, and keyframeless second-order variational regularization to recover smooth surfaces. Given an image sequence I_k from a moving camera with known pose $\mathbf{T}_k^W$, our task entails:

- Estimating the inverse depth for a set of sampled pixels (Section 3.1)
- Constructing the mesh using the sampled points (Section 3.2)
- Defining a suitable smoothness cost over the graph induced by the mesh (Section 3.3)
- Minimizing the smoothness cost (Section 3.4)
- Projecting the mesh from frame to frame (Section 3.5)

See Figure 3 for a block diagram of the data flow.

3.1. Feature Inverse Depth Estimation

We first estimate the inverse depth for a set "trackable" pixels (or *features*) sampled over the image domain that will serve as candidate vertices to insert into our mesh. Let $\mathcal{F}_k$ denote the current set of features. Each feature $f \in \mathcal{F}_k$ is detected at timestep f_t and defined at location $f_{\mathbf{u}} \in \Omega$ in the image I_{f_t} at pose $\mathbf{T}_{f_t}^W$.

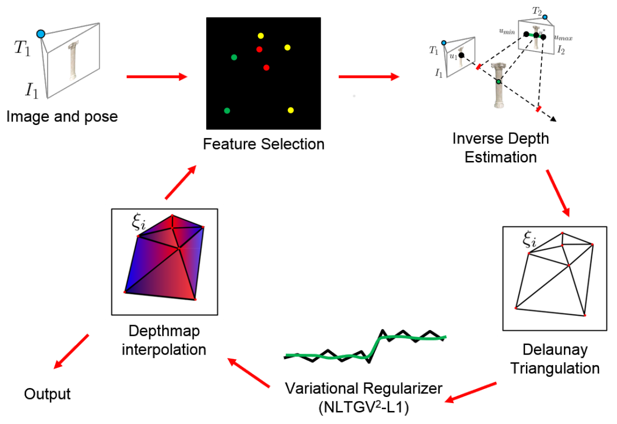

Figure 3: *FLaME Overview:* FLaME operates on image streams with known poses. Inverse depth is estimated for a set of *features* using the fast, filtering-based approach of [8]. When the inverse depth estimate for a given feature converges, it is inserted as a new vertex in a graph defined in the current frame and computed through Delaunay triangulations. This Delaunay graph is then used to efficiently smooth away noise in the inverse depth values and promote piecewise planar structure by minimizing a second-order variational cost defined over the graph.

We select features by dividing Ω into grid cells of size $2^L \times 2^L$ based on a user-set detail level L (see Figure 4) and selecting a pixel in each cell as a new feature if certain criteria are met. First, we do not select features in cells that contain the projection of another feature (this ensures we maintain a certain desired detail level). If no other feature in $\mathcal{F}_k$ falls into a given grid cell, then for each pixel $\mathbf{u}$ in the grid cell we compute a *trackability score* $s(\mathbf{u}) = |\nabla I_k(\mathbf{u})^T \mathbf{e}_\mathbf{u}|$ based on the image gradient $\nabla I_k(\mathbf{u})$ and epipolar direction $\mathbf{e}_\mathbf{u}$ induced by the previous frame. This score is a simple metric for determining pixels that will be easy to match in future frames given the camera motion. If the pixel in the window with the maximum score passes a threshold, we add it as a feature to $\mathcal{F}_k$.

Next, we estimate an inverse depth mean ξ_f and variance σ_f^2 for each $f \in \mathcal{F}_k$ by matching a reference patch of pixels around $\mathbf{u}_f$ in future frames using a direct search along the epipolar line. For a given match, we compute an inverse depth *measurement* with mean ξ_z and variance σ_z^2 according to the noise model in [8] and fuse it with the feature's current estimate using standard Bayesian fusion:

$$\xi_f \leftarrow \frac{\xi_f \sigma_z^2 + \xi_z \sigma_f^2}{\sigma_f^2 + \sigma_z^2}, \quad \sigma_f^2 \leftarrow \frac{\sigma_f^2 \sigma_z^2}{\sigma_f^2 + \sigma_x^2}. \tag{7}$$

3.2. Mesh Construction

We construct our mesh using the set of features $\mathcal{F}_k^* \subseteq \mathcal{F}_k$ whose inverse depth variance is lower than a threshold σ_{max}^2: $\mathcal{F}_k^* = \{f \in \mathcal{F}_k : \sigma_f^2 < \sigma_{max}^2\}$. We project these features into the current camera frame $\mathbf{T}_k^w$ and then compute a 2D Delaunay triangulation of the projected pixel locations using the fast method of [23, 24]. We denote the Delaunay triangulation by $\mathcal{DT}(\mathcal{F}_k^*) = (\mathcal{V}_k, \mathcal{T}_k)$, where $\mathcal{V}_k$ is the set of mesh vertices and $\mathcal{T}_k$ is the set of triangles. The Delaunay triangulation is optimal in the sense that it maximizes the minimum angle for each triangle in $\mathcal{T}_k$

We denote the feature corresponding to vertex $v \in \mathcal{V}_k$ with $v_f \in \mathcal{F}_k^*$ and let $v_\mathbf{u} \in \Omega$ denote the pixel location of v, which we initialize with the projection of f into the current frame k:

$$v_\mathbf{u} = \pi\left(\mathbf{K}\mathbf{T}_{f_t}^k \pi^{-1}\left(f_\mathbf{u}, \xi_f^{-1}\right)\right). \tag{8}$$

Finally, we assign an inverse depth to each new vertex that we refer to as v_z and initialize it to the feature inverse depth ξ_f for corresponding feature f projected into the current frame. Note that although we perform our triangulation in 2D using the vertex pixel locations, we can project the mesh to 3D using this inverse depth value. We can also obtain a dense inverse depthmap $\xi : \Omega \to \mathbb{R}_+$ by linearly interpolating the inverse depth values of the mesh vertices.

3.3. Non-Local Second Order Variational Cost

Now equipped with an inverse depth mesh $\mathcal{DT}(\mathcal{F}_k^*)$, we formulate our non-local, graph-based variational regularizer that will efficiently smooth away noise in the mesh and promote planar structure.

We start with the continuous NLTGV^2-L_1 variational cost for a fully dense inverse depthmap $\xi : \Omega \to \mathbb{R}_+$, which sets the smoothing term $E_{smooth}(\xi)$ to NLTGV^2 defined in Equation (4) and the data fidelity term $E_{data}(\xi)$ to the robust L_1 norm in Equation (2):

$$E(\xi) = \text{NLTGV}^2(\xi) + \lambda \int_\Omega |\xi(\mathbf{u}) - z(\mathbf{u})| \, d\mathbf{u}. \tag{9}$$

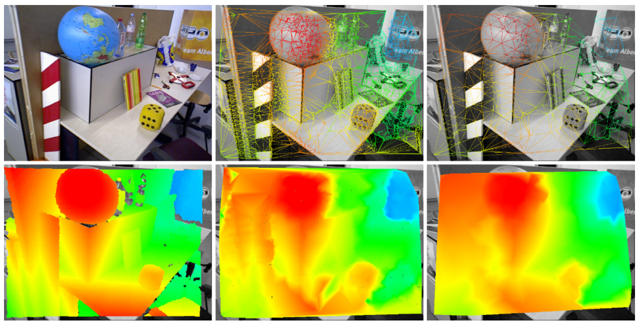

Figure 4: *Detail vs. Speed:* The density of tracked features can be tuned to favor geometric detail or speed. Here we compare the depthmaps (bottom row, columns 2 and 3) generated from the smoothed graph (top row, columns 2-3) to a Kinect depthmap (bottom left) using different settings of detail parameter L (see Section 3.1). The input RGB image is shown in the top left and the depthmaps are colored by depth. Changing L from 4 (column 2) to 5 (column 3) results in nearly a 50 percent speedup (see Table 1).

Here $z : \Omega \rightarrow \mathbb{R}_+$ is our raw, unsmoothed inverse depthmap.

We will approximate this functional over the fully dense ξ using our inverse depth mesh $\mathcal{DT}(\mathcal{F}_k^*)$. We first reinterpret $\mathcal{DT}(\mathcal{F}_k^*)$ as a directed *Delaunay graph* $\mathcal{DG}(\mathcal{F}_k^*) = (\mathcal{V}_k, \mathcal{E}_k)$, with identical vertices $\mathcal{V}_k$ and directed, non-parallel edges $\mathcal{E}_k$ generated from the triangle set $\mathcal{T}_k$ (the direction of each edge is arbitrary). For each vertex $v \in \mathcal{V}_k$, we assign a smoothed inverse depth value that we denote v_ξ and an auxiliary variable $\mathbf{w} \in \mathbb{R}^2$ such that $v_\mathbf{w} = (v_{w_1}, v_{w_2})$. We let $v_\mathbf{x}$ denote $(v_\xi, v_\mathbf{w})$.

The graph version of our L_1 data fidelity term is straightforward to define in terms of $\mathcal{DG}(\mathcal{F}_k^*)$ by replacing the integral over the image domain Ω with a sum over the vertices of $\mathcal{DG}(\mathcal{F}_k^*)$:

$$\int_\Omega |\xi(\mathbf{u}) - z(\mathbf{u})| \, d\mathbf{u} \approx \sum_{v \in \mathcal{V}_k} |v_\xi - v_z| \, , \tag{10}$$

where v_z is the inverse depth of feature v_f projected into the current frame.

Discretizing the NLTGV^2 smoothing term over $\mathcal{DG}(\mathcal{F}_k^*)$ simply requires a special setting of the weight functions α, $\beta : \Omega \times \Omega \rightarrow \mathbb{R}$. In the non-local, variational framework, these functions control the influence of inverse depth values over their spatial neighbors and thus should be defined in terms of the edge set $\mathcal{E}_k$ of $\mathcal{DG}(\mathcal{F}_k^*)$.

For each edge $e \in \mathcal{E}_k$, we denote the associated vertices as $v^i, v^j \in \mathcal{V}_k$ (note again that the edges are directed from i to j). We then assign weights $e_\alpha, e_\beta \geq 0$ to each edge and set the functions α and β to the following:

$$\alpha(\mathbf{u}, \mathbf{v}) = e_\alpha \delta(\mathbf{u} - v^i_\mathbf{u}, \mathbf{v} - v^j_\mathbf{u}) \quad \text{for } e \in \mathcal{E} \tag{11}$$

$$\beta(\mathbf{u}, \mathbf{v}) = e_\beta \delta(\mathbf{u} - v^i_\mathbf{u}, \mathbf{v} - v^j_\mathbf{u}) \quad \text{for } e \in \mathcal{E}. \tag{12}$$

Setting α, β in terms of delta functions that encode the connectivity in the graph (i.e. the edges $\mathcal{E}_k$) reduces the double integral over Ω in Equation 4 to a summation over $\mathcal{E}_k$:

$$\begin{aligned} \text{NLTGV}^2(\xi) \approx & \sum_{e \in \mathcal{E}_k} e_\alpha \left| v^i_\xi - v^j_\xi - \langle v^i_\mathbf{w}, v^i_\mathbf{u} - v^j_\mathbf{u} \rangle \right| \quad + \\ & \sum_{e \in \mathcal{E}_k} e_\beta \left| v^i_{w_1} - v^j_{w_1} \right| \quad + \\ & \sum_{e \in \mathcal{E}_k} e_\beta \left| v^i_{w_2} - v^j_{w_2} \right| \\ = & \sum_{e \in \mathcal{E}_k} \left|\left| \mathbf{D}_e(v^i_\mathbf{x}, v^j_\mathbf{x}) \right|\right|_1 . \end{aligned} \tag{13}$$

Here $\mathbf{D}_e$ is a linear operator that acts on the vertices corresponding to edge e:

$$\mathbf{D}_e(v^i_\mathbf{x}, v^j_\mathbf{x}) = \begin{bmatrix} e_\alpha \left(v^i_\xi - v^j_\xi - \langle v^i_\mathbf{w}, v^i_\mathbf{u} - v^j_\mathbf{u} \rangle \right) \\ e_\beta \left(v^i_{w_1} - v^j_{w_1} \right) \\ e_\beta \left(v^i_{w_2} - v^j_{w_2} \right) \end{bmatrix} . \tag{14}$$

The final form of our graph-based NLTGV$^2 - L_1$ cost functional is now

$$\begin{aligned} & E(\mathcal{DG}(\mathcal{F}_k^*)) = \\ & \quad \sum_{e \in \mathcal{E}_k} \left|\left| \mathbf{D}_e(v^i_\mathbf{x}, v^j_\mathbf{x}) \right|\right|_1 + \lambda \sum_{v \in \mathcal{V}_k} |v_\xi - v_z| \, . \end{aligned} \tag{15}$$

Note that by defining the NLTGV^2-L_1 variational cost in terms of the $\mathcal{DG}(\mathcal{F}_k^*)$, we can trivially augment and refine the objective by simply adding new vertices and edges to the graph, just as the mesh $\mathcal{DT}(\mathcal{F}_k^*)$ is augmented and refined using incremental triangulations.

3.4. Graph Optimization

Having reformulated the NLTGV^2-L_1 cost in terms of graph $\mathcal{DG}(\mathcal{F}_k^*)$, we now apply the optimization method of Chambolle and Pock [4]. We see the summation over $\mathcal{E}_k$ and the summation over $\mathcal{V}_k$ in Equation 15 correspond to $F(\mathbf{D}(\mathbf{x}))$ and $G(\mathbf{x})$, respectively, in the Chambolle and Pock objective in Equation 5 for $\mathbf{x} = [v_\mathbf{x}]$ for $v \in \mathcal{V}_k$, and that we can follow the same optimization approach.

We first generate the saddlepoint problem induced by this new graph-based cost by re-expressing the L_1 norm corresponding to the sum over edges in terms of its convex conjugate, which can in turn be composed of the conjugates of each term in the sum:

$$\left|\left| \mathbf{D}_e(v^i_\mathbf{x}, v^j_\mathbf{x}) \right|\right|_1 = \max_{e_\mathbf{q}} \langle \mathbf{D}_e(v^i_\mathbf{x}, v^j_\mathbf{x}), e_\mathbf{q} \rangle - \delta_Q(e_\mathbf{q}), \tag{16}$$

where we have assigned a dual variable $\mathbf{q} \in \mathbb{R}^3$ to each edge, denoted by $e_\mathbf{q}$. The indicator term δ_Q is the conjugate L_1^* and is defined as

$$\delta_Q(e_\mathbf{q}) = \sum_{i=1}^{3} \delta_{q_i}(e_{q_i}), \quad \delta_q(q) \quad = \begin{cases} 0 & \text{if } |q| \leq 1 \\ \infty, & \text{otherwise} \end{cases} . \tag{17}$$

The NLTGV^2-L_1 saddlepoint problem can now be written in terms of $\mathcal{DG}(\mathcal{F}_k^*)$ as:

$$\begin{aligned} \min_{v_\mathbf{x}} \max_{e_\mathbf{q}} & \sum_{e \in \mathcal{E}_k} \langle \mathbf{D}_e(v^i_\mathbf{x}, v^j_\mathbf{x}), e_\mathbf{q} \rangle - \delta_Q(e_\mathbf{q}) \\ & + \lambda \sum_{v \in \mathcal{V}_k} |v_\xi - v_z| \, . \end{aligned} \tag{18}$$

To optimize Equation 18, we first perform semi-implicit, subgradient ascent over $e_\mathbf{q}$ for each $e \in \mathcal{E}_k$:

$$e^{n+1}_\mathbf{q} = e^n_\mathbf{q} + \sigma \mathbf{D}_e(v^i_\mathbf{x}, v^j_\mathbf{x}) - \sigma \partial \delta_Q(e^{n+1}_\mathbf{q}) \tag{19}$$

where $\sigma > 0$ is the dual step size and $\partial \delta_Q(\mathbf{q})$ is the subgradient of $\delta_Q(\mathbf{q})$. Moving the $e^{n+1}_\mathbf{q}$ terms to the left side of the equation yields:

$$e^{n+1}_\mathbf{q} + \sigma \partial \delta_Q(e^{n+1}_\mathbf{q}) = e^n_\mathbf{q} + \sigma \mathbf{D}_e(v^i_\mathbf{x}, v^j_\mathbf{x}), \tag{20}$$

Algorithm 2 NLTGV$^2 - L_1$ Graph Optimization

```
// Choose $\sigma, \tau > 0$, $\theta \in [0, 1]$.
while not converged do
    for each $e \in \mathcal{E}_k$ do
        $e_{\mathbf{q}}^{n+1} = \text{prox}_{F^*}\left(e_{\mathbf{q}}^n + \sigma \mathbf{D}_e(v_{\bar{\mathbf{x}}}^i, v_{\bar{\mathbf{x}}}^j)\right)$
    for each $v \in \mathcal{V}_k$ do
        $v_{\mathbf{x}}^{n+1} = \text{prox}_G\left(v_{\mathbf{x}}^n - \tau \sum_{e \in \mathcal{N}_{in}(v)} \mathbf{D}_{in}^*(e_{\mathbf{q}}^{n+1})\right.$
                $\left. - \tau \sum_{e \in \mathcal{N}_{out}(v)} \mathbf{D}_{out}^*(e_{\mathbf{q}}^{n+1})\right)$
        $v_{\bar{\mathbf{x}}}^{n+1} = v_{\mathbf{x}}^{n+1} + \theta\left(v_{\mathbf{x}}^{n+1} - v_{\mathbf{x}}^n\right)$
```

which we can express in terms of the proximal operator:

$$e_{\mathbf{q}}^{n+1} = \text{prox}_{F^*}\left(e_{\mathbf{q}}^n + \sigma \mathbf{D}_e(v_{\mathbf{x}}^i, v_{\mathbf{x}}^j)\right). \qquad (21)$$

We next wish to perform semi-implicit subgradient *descent* over $v_{\mathbf{x}}$ for each $v \in \mathcal{V}_k$, but some care must be taken with forming the adjoint operator $\mathbf{D}_e^*$. We observe that the operator $\mathbf{D}_e(v_{\mathbf{x}}^i, v_{\mathbf{x}}^j)$ maps two primal vertex variables (corresponding to the source and destination vertex) to the dual space for each edge. The adjoint must therefore map a single dual edge variable to the space of of two primal variables (again corresponding to the source and destination vertex). Starting from the expression of $\mathbf{D}_e$ in Equation 14, we form the adjoint as

$$\mathbf{D}_e^*(e_{\mathbf{q}}) = \begin{bmatrix} e_\alpha e_{q_1} \\ -e_\alpha(e_u^i - e_u^j)e_{q_1} + e_\beta e_{q_2} \\ -e_\alpha(e_v^i - e_v^j)e_{q_1} + e_\beta e_{q_3} \\ -e_\alpha e_{q_1} \\ -e_\beta e_{q_2} \\ -e_\beta e_{q_3} \end{bmatrix} \qquad (22)$$

$$= \begin{bmatrix} \mathbf{D}_{in}^*(e_{\mathbf{q}}) \\ \mathbf{D}_{out}^*(e_{\mathbf{q}}) \end{bmatrix}, \qquad (23)$$

where we have partitioned the top three rows $\mathbf{D}_e^*$ into $\mathbf{D}_{in}^*$, which maps $e_{\mathbf{q}}$ to the source primal vertex space, and the bottom three rows of $\mathbf{D}_e^*$ into $\mathbf{D}_{out}^*$, which maps $e_{\mathbf{q}}$ to the destination primal vertex space.

The semi-implicit subgradient descent equations for each $v_{\mathbf{x}}$ is then given by

$$v_{\mathbf{x}}^{n+1} = v_{\mathbf{x}}^n - \tau \partial G(v_{\mathbf{x}}^{n+1}) - \tau \sum_{e \in \mathcal{N}_{in}(v)} \mathbf{D}_{in}^*(e_{\mathbf{q}}^{n+1}) - \tau \sum_{e \in \mathcal{N}_{out}(v)} \mathbf{D}_{out}^*(e_{\mathbf{q}}^{n+1}), \qquad (24)$$

where $\tau > 0$ is the primal step size and the incoming and outgoing edges of v are denoted as $\mathcal{N}_{in}(v)$ and $\mathcal{N}_{out}(v)$, respectively. Solving for $v_{\mathbf{x}}^{n+1}$ in terms of the proximal operator of G then yields

$$v_{\mathbf{x}}^{n+1} = \text{prox}_G\left(v_{\mathbf{x}}^n - \tau \sum_{e \in \mathcal{N}_{in}(v)} \mathbf{D}_{in}^*(e_{\mathbf{q}}^{n+1}) - \tau \sum_{e \in \mathcal{N}_{out}(v)} \mathbf{D}_{out}^*(e_{\mathbf{q}}^{n+1})\right). \qquad (25)$$

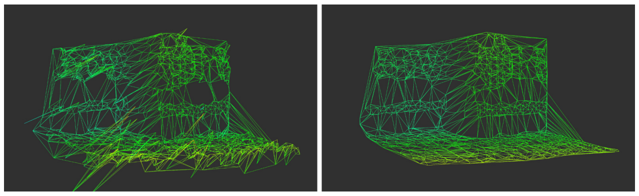
Figure 5: *Spatial Regularization:* FLaME minimizes a non-local, variational smoothness cost defined over a Delaunay graph, which efficiently generates piecewise planar mesh reconstructions from noisy inverse depth estimates. The above images show the meshes produced from raw, unsmoothed inverse depth values (left) and those smoothed with FLaME (right).

The final step of the Chambolle and Pock method is a simple extragradient step applied to each vertex. The full optimization is summarized in Algorithm 2. By expressing the optimization in terms of the graph $\mathcal{DG}(\mathcal{F}_k^*)$, we can trivially add and remove vertices and edges to the objective as new features are added to $\mathcal{F}_k^*$ and triangulated. In addition, by matching the density of vertices to the geometric complexity of the observed environment, each optimization iteration is both fast to perform and quick to converge (see Figure 5 for a comparison between smoothed and unsmoothed meshes). The graph interpretation also provides additional intuition into the optimization, which alternates between operations on the vertices and edges of the graph: smoothing updates are passed from vertices to neighboring edges, and then from the edges back to the corresponding vertices.

3.5. Frame-to-Frame Propagation

We propagate the Delaunay graph $\mathcal{DG}(\mathcal{F}_k^*)$ so that the optimized surface is always available in the current frame. At each timestep, we set the vertex location $v_{\mathbf{u}}$ for $v \in \mathcal{V}_k$ to the projection based on the smoothed inverse depth value v_ξ, which we then update to be expressed in the current frame as well. We also set the unsmoothed v_z inverse depth value to the projection of the underlying feature inverse depth ξ_f.

With new vertex locations and potentially new features, we retriangulate to maintain the Delaunay optimality property and add and remove edges to reflect the new connectivity in the triangulation. We also remove vertices and features that project outside the view of the current camera, although these could be saved and displayed separately. Since the relative transform from frame-to-frame is small for high framerate cameras, and because the optimization is typically able to converge before the next frame is available, the optimization is relatively unaffected by the projection step and we benefit from the smoothed surface always available at the current camera.

4. Evaluation

Our implementation of FLaME is written in C++ and makes use of the Boost Graph Library [25] and the Delaunay triangulation implementation from [23, 24]. The primary processing thread handles stereo matching and inverse depth filtering, performs Delaunay triangulations, updates the Delaunay graph, and publishes output. A second thread continuously performs the graph optimization steps outlined in Algorithm 2. The third thread samples new features every N frames ($N = 6$ in our experiments). For all experiments we set the edge weight $e_\alpha = 1/||v^i_{\mathbf{u}} - v^j_{\mathbf{u}}||_2$ (the reciprocal of the edge length in pixels) and $e_\beta = 1$. We set the parameter λ that controls the balance between unsmoothed data and the regularizer between $[0.1, 0.35]$.

4.1. Benchmark Datasets

We quantitatively compare the FLaME reconstructions to existing approaches and show how we are able to produce accurate, dense geometry extremely efficiently at the current frame. We interpolate the FLaME meshes to fully dense inverse depthmaps and measure their accuracy and completeness against two existing CPU-based approaches: LSD-SLAM [6] and Multi-Level Mapping (MLM) [11]. We use image and pose sequences from the TUM RGB-D SLAM Benchmark dataset (VGA at 30 Hz) [27] and the EuRoC MAV datasets (WVGA at 20 Hz) [3]. Pose ground truth for both datasets was generated using a motion capture system. Structure ground truth was approximated using an RGB-D sensor for the TUM sequences and a 3D laser scanner for the EuRoC sequences.

The pose tracking and SLAM backend modules of LSD-SLAM and MLM were disabled in the experiments so that all three algorithms used the motion capture poses and all performance differences can be attributed to the different depth estimation techniques.

All metrics were captured on a desktop computer with an Intel Core i7-4820K 3.7 GHz CPU. We use three sequences from the TUM dataset (long_office_household, structure_texture_far, and nostructure_texture_near_withloop) and the 6 sequences from the EuRoC dataset with structure ground truth (V1_01, ..., V2_03).

We report two main measures for depthmap accuracy and completeness: the *relative inverse depth error* and the *density of accurate depth estimates*. The relative inverse depth error is the error in inverse depth relative to ground truth, averaged over all pixels and depthmaps. The density of accurate depth estimates is the fraction of inverse depth estimates that are within 10 percent of ground truth for each depthmap, averaged over all depthmaps. We also report both runtime per frame and CPU load over the datasets.

Performance on Benchmark Datasets

		DM	RE [%]	AD [%]	Cores	Time [ms]	FPS [Hz]
TUM	LSD	181	18	19	2.5	16	62
	MLM	103	12	32	2.1	17	57
	L=3	**4950**	8.5	**54**	2.0	16	61
	L=4	**4950**	6.8	**54**	1.7	7.3	136
	L=5	**4950**	**6.6**	51	**1.4**	**4.2**	**236**
EuRoC	LSD	874	18	17	1.6	16	61
	MLM	734	17	25	1.0	14	69
	L=3	**12595**	12	36	1.3	13	78
	L=4	**12595**	11	**37**	1.2	7.0	143
	L=5	**12595**	**10**	33	**0.8**	**4.3**	**230**

Table 1: We evaluate FLaME with various settings of detail parameter L (see Section 3.1) on two benchmark datasets [27, 3]. FLaME produces depthmaps (DM) with both lower relative inverse depth error (RE) and a higher density of accurate inverse depths (AD), while taking less processing time per frame (Time) and using less CPU load (Cores), than state-of-the-art approaches LSD-SLAM [6] and MLM [11]. Refer to Section 4.1 for a more detailed description of the metrics and experimental setup.

Accurate Inverse Depth Density [%]

	Sequence	LSD	MLM	L=3	L=4	L=5
TUM	`fr3_loh`	18.9	30.4	**48.1**	47.9	44.1
	`fr3_nstn`	16.5	30.6	52.5	**53.7**	51.9
	`fr3_stf`	26.8	47.8	**72.6**	72.0	69.1
EuRoC	`V1_01`	18.2	26.4	36.6	**37.8**	34.4
	`V1_02`	18.3	27.9	39.2	**40.5**	37.8
	`V1_03`	11.0	17.0	26.4	**27.0**	24.7
	`V2_01`	25.9	39.3	**48.1**	46.9	41.6
	`V2_02`	20.5	28.9	**37.2**	**37.2**	32.2
	`V2_03`	11.5	19.0	27.8	**28.9**	26.7

Table 2: Here we present the fraction of inverse depths per depthmap that are within 10 percent of groundtruth for each benchmark video sequence [27, 3] for LSD-SLAM [6], MLM [11], and FLaME with different settings of parameter L. FLaME outperforms the competing approaches, with $L = 4$ providing a nice balance between the number of vertices per depthmap and the amount of smoothing performed.

The results are summarized in Table 1 and Table 2 and as can be seen, FLaME produces dense geometry more accurately and efficiently than the competing approaches. On the EuRoC sequences FLaME with $L = 5$ produces reconstructions at up to 230 Hz using less than one Intel i7 CPU core and achieves the lowest relative inverse depth error across the different systems. Although finer settings of $L = 3$ and $L = 4$ produce slightly better density metrics, as expected, they fair slightly worse in terms of relative inverse depth error compared to $L = 5$. We believe the primary reason for this unintuitive result is that the graph optimization takes longer to converge for these parameter settings given the greater number of vertices and edges. Since the mesh

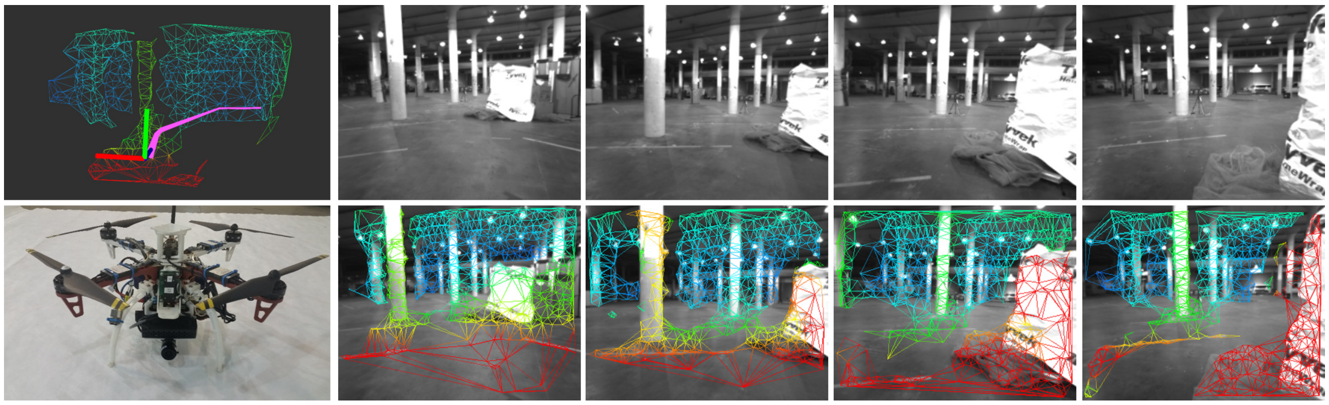

Figure 6: *Flight Experiments:* FLaME can be used to enable online perception for autonomous navigation. We conducted indoor and outdoor flight experiments running FLaME onboard a small micro-aerial vehicle (MAV) (bottom left) with a forward-facing camera flying at up to 3.5 meters-per-second. The image on the top left shows the collision-free trajectory (pink) that is generated to navigate around a pillar obstacle. The images to the right show the inverse depth meshes as the vehicle approaches the obstacle field.

takes longer to converge, the camera moves before the mesh can settle, resulting in higher inverse depth error. Initial depthmap convergence is very fast, however, usually within the first second of operation. In addition, FLaME produces accurate meshes at the current frame, while the competing approaches (which are both keyframe based) produce reconstructions far more infrequently, which is particularly dangerous for mobile robot navigation.

We also experimented with corrupting the ground truth positions with additive Gaussian noise to characterize the effect of pose error. With no artificial noise, the density of accurate inverse depths for the TUM `fr3_stf` sequence is 71 percent. However, with translation noise with a standard deviation of 1 cm, this density drops to 30 percent, which demonstrates the importance of accurate pose information on the depth estimation process. Addressing this limitation is one direction that we would like to pursue for future work.

4.2. Flight Experiments

We also provide results from experiments with FLaME running completely onboard, in-the-loop on a small autonomous quadrotor flying at up to 3.5 meters-per-second.

Timing and Load on Autonomous MAV

Metric	Indoor	Outdoor
Vehicle Speed [m/s]	2.5	3.5
Depthmaps	803	1046
CPU Load [cores]	1.6	1.7
Runtime [ms]	9.4	11
Peak FPS [Hz]	106	91

Table 3: FLaME is efficient enough to allow for real-time perception onboard small, computationally constrained micro-aerial vehicles (MAVs). We flew our quadrotor fully autonomously in both indoor and outdoor environments with no prior information and used geometry from FLaME to plan around obstacles online.

The quadrotor (see Figure 6) weighed 3 kg and was equipped with a Point Grey Flea 3 camera running at 320×256 image resolution at 60 Hz, an inertial measurement unit (IMU), and a laser altimeter. The pose of the camera was provided by an external visual-inertial odometry pipeline [26]. Collision-free motion plans were generated using A* on a 2D occupancy grid updated using slices from the FLaME meshes. All computation was performed onboard an Intel Skull Canyon NUC flight computer, with no prior information provided to the robot.

We flew the vehicle through an indoor warehouse environment and an outdoor forest with obstacles that the vehicle had to plan around using perception from FLaME. Runtime and load metrics on the flight computer are summarized in Table 3. Even on the flight computer, FLaME was still able to produce dense reconstructions at over 90 Hz (the detail parameter was set to $L = 3$), with sufficient accuracy to plan around obstacles.

5. Conclusion

We presented a novel dense monocular depth estimation algorithm capable of reconstructing geometric meshes on computationally constrained platforms. FLaME reformulates the reconstruction problem as a variational smoothing problem over a time-varying Delaunay graph, which allows both for efficient, incremental smoothing of noisy depth estimates and low-latency mesh estimation.

Acknowledgments

This material is based upon work supported by DARPA under Contract No. HR0011-15-C-0110 and by the NSF Graduate Research Fellowship under Grant No. 1122374. We also thank John Ware, John Carter, and the rest of the MIT-Draper FLA Team for continued development and testing support.

References

[1] H. Alvarez, L. M. Paz, J. Sturm, and D. Cremers. Collision avoidance for quadrotors with a monocular camera. In *Experimental Robotics*, pages 195–209. Springer, 2016.

[2] K. Bredies, K. Kunisch, and T. Pock. Total generalized variation. *Journal on Imaging Sciences*, 2010.

[3] M. Burri, J. Nikolic, P. Gohl, T. Schneider, J. Rehder, S. Omari, M. W. Achtelik, and R. Siegwart. The EuRoC micro aerial vehicle datasets. 2016.

[4] 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.

[5] S. Daftry, S. Zeng, A. Khan, D. Dey, N. Melik-Barkhudarov, J. A. Bagnell, and M. Hebert. Robust monocular flight in cluttered outdoor environments. *arXiv preprint arXiv:1604.04779*, 2016.

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

[7] J. Engel, J. Sturm, and D. Cremers. Camera-based navigation of a low-cost quadrocopter. In *Proc. IROS*, 2012.

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

[9] C. Forster, L. Carlone, F. Dellaert, and D. Scaramuzza. Imu preintegration on manifold for efficient visual-inertial maximum-a-posteriori estimation. In *RSS*, 2015.

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

[11] W. N. Greene, K. Ok, P. Lommel, and N. Roy. Multi-Level Mapping: Real-time Dense Monocular SLAM. In *Proc. ICRA*, 2016.

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

[13] A. I. Mourikis and S. I. Roumeliotis. A multi-state constraint kalman filter for vision-aided inertial navigation. In *Proc. ICRA*, 2007.

[14] R. Mur-Artal, J. Montiel, and J. D. Tardos. Orb-slam: a versatile and accurate monocular slam system. *Trans. on Robotics*, 2015.

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

[16] P. Ondruska, P. Kohli, and S. Izadi. MobileFusion: Real-time volumetric surface reconstruction and dense tracking on mobile phones. In *Trans. on Visualization and Computer Graphics*, 2015.

[17] N. Parikh, S. P. Boyd, et al. Proximal algorithms. *Foundations and Trends in Optimization*, 2014.

[18] S. Pillai, S. Ramalingam, and J. J. Leonard. High-Performance and Tunable Stereo Reconstruction. In *Proc. ICRA*, 2016.

[19] P. Piniés, L. M. Paz, and P. Newman. Dense mono reconstruction: Living with the pain of the plain plane. In *Proc. ICRA*, 2015.

[20] R. Ranftl, K. Bredies, and T. Pock. Non-local total generalized variation for optical flow estimation. In *Proc. ECCV*. Springer International Publishing, 2014.

[21] T. Schöps, J. Engel, and D. Cremers. Semi-dense visual odometry for ar on a smartphone. In *Proc. ISMAR*. IEEE, 2014.

[22] T. Schöps, T. Sattler, C. Häne, and M. Pollefeys. 3d modeling on the go: Interactive 3d reconstruction of large-scale scenes on mobile devices. In *3D Vision (3DV)*, 2015.

[23] J. R. Shewchuk. Triangle: Engineering a 2D Quality Mesh Generator and Delaunay Triangulator. In *Applied Computational Geometry: Towards Geometric Engineering*, 1996.

[24] J. R. Shewchuk. Delaunay refinement algorithms for triangular mesh generation. *Computational Geometry*, 2002.

[25] J. G. Siek, L.-Q. Lee, and A. Lumsdaine. *The Boost Graph Library: User Guide and Reference Manual, Portable Documents*. Pearson Education, 2001.

[26] T. J. Steiner, R. D. Truax, and K. Frey. A vision-aided inertial navigation system for agile high-speed flight in unmapped environments. In *Proce. IEEE Aerospace Conference*, 2017.

[27] 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.

[28] A. Wendel, M. Maurer, G. Graber, T. Pock, and H. Bischof. Dense reconstruction on-the-fly. In *Proc. CVPR*, 2012.