

Robust Real-Time Visual Odometry for Dense RGB-D Mapping

Thomas Whelan¹, Hordur Johannsson², Michael Kaess², John J. Leonard² and John McDonald¹

Abstract—This paper describes extensions to the Kintinuous [1] algorithm for spatially extended KinectFusion, incorporating the following additions: (i) the integration of multiple 6DOF camera odometry estimation methods for robust tracking; (ii) a novel GPU-based implementation of an existing dense RGB-D visual odometry algorithm; (iii) advanced fused real-time surface coloring. These extensions are validated with extensive experimental results, both quantitative and qualitative, demonstrating the ability to build dense fully colored models of spatially extended environments for robotics and virtual reality applications while remaining robust against scenes with challenging sets of geometric and visual features.

I. INTRODUCTION

The advent of the Microsoft Kinect and other RGB-D sensors has resulted in great progress in dense mapping and SLAM in recent years [2], [3], [4]. Given the low cost of the sensor coupled with the large scale availability of GPUs for high performance computing, dense methods are becoming more popular in tackling some of the key perception problems in robotics [5], [6], [7]. The KinectFusion algorithm in particular, introduced by Newcombe *et al.*, was one of the first systems to produce a volumetric reconstruction of a scene in real-time with an unprecedented level of accuracy [6]. While a volumetric representation is useful for planning robotic tasks such as manipulation, this algorithm has a number of limitations. In our previous work we extended KinectFusion to function over an extended area [1].

A notable feature of the KinectFusion algorithm is use of depth information alone for camera motion tracking. The underlying odometry estimation algorithm, iterative closest point (ICP), is prone to failure in situations where camera displacement is large between frames or a lack of 3D depth features poorly constrains the camera pose in the observed scene. For example, camera tracking performance will suffer when pointed at a flat wall or corridor with no significant 3D features present. In our previous paper we presented some preliminary work on remedying this problem by means of incorporating a visual odometry method for camera pose estimation in the KinectFusion pipeline [1].

We present results demonstrating that the combination of various odometry estimation techniques increases the robustness of camera tracking across a variety of environments, from desk sized manipulation type environments to corridors and extended scale paths. We also present a novel GPU-implementation of the RGB-D-based visual odometry system

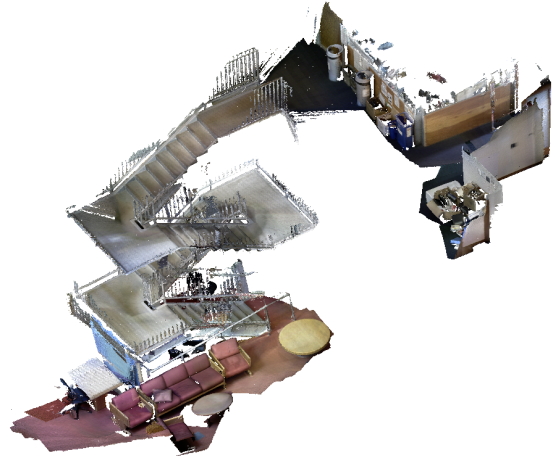


Fig. 1. Triangular mesh stairwell produced in real-time by the Kintinuous system containing over 2.6 million triangles and 1.5 million colored vertices.

of Steinbruecker *et al.* [8], enabling real-time execution of the algorithm. Additionally we present a method for intelligently integrating RGB color information into the KinectFusion reconstruction process to allow high quality fully colored map production. The method we present results in real-time colored volumetric surface reconstructions without the use of keyframes. Although the original KinectFusion algorithm was published with images and videos showing colored surface reconstructions, this method was not documented and only ever described as “texture mapped” [9].

II. RELATED WORK

A number of different approaches have been used to solve the odometry estimation problem in RGB-D-based mapping systems. Visual feature matching using various keypoint descriptors for pose estimation have been popular in SLAM systems [10], [11], [2]. As discussed in the previous section, the KinectFusion system relies purely on dense ICP every frame to determine the camera pose. Henry *et al.* opted to combine ICP with visual features for a more robust pose estimate. Less commonly an image warping method is used in the parameterisation of a camera transformation as used by Audras *et al.* [4] and Steinbruecker *et al.* [8].

Since the release of KinectFusion in 2011 a number of derived works have followed, notably our own system Kintinuous [1], the commercial ReconstructMe product [12], the KinFu Large Scale open source PCL project [13] and the Moving Volume KinectFusion work of Roth and Vona [14]. At the time of writing none of these works have incorporated alternative methods of camera tracking in place of ICP. In addition none of these works have shown results of real-time

¹T. Whelan and J. McDonald are with the Department of Computer Science, National University of Ireland Maynooth, Co. Kildare, Ireland. thomas.j.whelan at nuim.ie

²H. Johannsson, M. Kaess and J. J. Leonard are with Computer Science and Artificial Intelligence Laboratory (CSAIL), Massachusetts Institute of Technology (MIT), Cambridge, MA 02139, USA.

color integration. KinFu Large Scale and ReconstructMe have demonstrated post processed texture mapping.

In their future work Steinbruecker *et al.* mention implementing their RGB-D odometry algorithm on GPU [8]. Such work is yet to be presented. An open sourced CPU implementation does exist, released into the OpenCV contrib module in March 2012 [15]. Our GPU implementation is based off of this release. During preparation of this paper we became aware of concurrent work by Maria Dimashova of Itseez (author of the OpenCV contrib code) in combining the RGB-D odometry algorithm of Steinbruecker *et al.* with ICP, CPU-based however.

By far the most popular method for integrating color information into constructed maps is keyframe style colored point clouds [4], [2], while some approaches ignore color completely [11]. However some systems, such as work by Henry *et al.* and Stückler and Behnke, make use of surfels, angles of observation and multi-resolution color distributions for intelligent color selection [16], [17]. With the exception of these two approaches, none of the systems discussed utilise any kind of fused color integration and produce a map effectively colored by keyframe information. Such keyframe based coloring typically results in the incorporation of pronounced sensor noise in the mapped surface color.

III. BACKGROUND

The Kintinuous system is an extension of the KinectFusion algorithm published by Newcombe *et al.* in 2011 [6]. A core component of the KinectFusion algorithm is the truncated signed distance function (TSDF), a volumetric surface representation where each element stores the signed distance to the closest surface [18]. For data integration, KinectFusion computes a vertex and normal map from the raw depth map which is then used to compute the camera pose via an ICP-based registration with the predicted surface model raycast from the current TSDF. Given this pose, the depth data is integrated into the TSDF through a weighted running average which over time results in a smooth surface reconstruction. Kintinuous implements the TSDF as a cyclical buffer allowing the reconstructed area to move around the real world. Upon translation, any surface exiting the volume is extracted by raycasting the volume only in a small slice defined by a movement threshold, outputting a point cloud which is then triangulated to create a mesh surface, as shown in Figure 2.

In the following section we describe our high performance GPU implementation of Steinbruecker *et al.*'s RGB-D odometry algorithm allowing real-time operation within the Kintinuous framework. Following that in Section V we describe the combination of various odometry estimation algorithms and a method for combining the dense ICP and RGB-D odometry estimators to produce an estimator which makes use of both dense depth and photometric information.

IV. GPU-BASED RGB-D ODOMETRY

As discussed in Section I, a reliance on depth information alone for the estimation of the camera pose has a number of

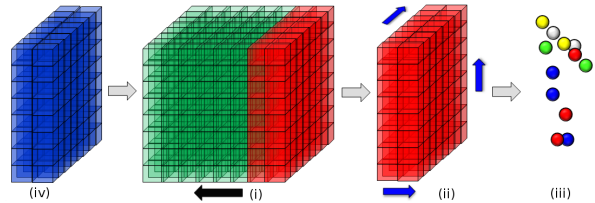


Fig. 2. The four main steps of the Kintinuous algorithm are shown above; (i) Camera motion exceeds movement threshold (black arrow); (ii) Volume slice (red) is raycast (orthogonal directions shown in blue arrows) for point extraction and reset; (iii) Point cloud extracted and fed into greedy mesh triangulation algorithm [19]; (iv) New spatial region enters volume (blue).

well understood problems. In order to remove the susceptibility to these problems we have developed a GPU implementation of an existing dense RGB-D-based visual odometry algorithm presented by Steinbruecker *et al.* and integrated it into the Kintinuous pipeline. Given that the original ICP odometry estimator uses dense information for camera pose estimation, we chose a visual odometry algorithm which also used a dense method over a sparse feature based approach. The original work by Steinbruecker *et al.* documented an execution speed on CPU of 12.5Hz [8]. Additionally, we measured the open source CPU implementation available in the OpenCV contrib module to operate at a speed of 20Hz on our test platform (specifications in Section VII-C) [15]. While alone these speeds are adequate for real-time visual odometry neither implementation matched the speed of the original KinFu ICP odometry estimator, which runs at over 100Hz on our test platform. For this reason we chose to reimplement the work by Steinbruecker *et al.* in CUDA for massively parallel operation on a GPU.

A. CUDA Implementation

Given two consecutive RGB image and depth map pairs $[\mathbf{I}_{RGB_n}, \mathbf{D}_n]$ and $[\mathbf{I}_{RGB_{n+1}}, \mathbf{D}_{n+1}]$ with $\mathbf{I}_{RGB}(x, y) \in \mathbb{N}^3$ and $\mathbf{D}(x, y) \in \mathbb{N}$ we compute a rigid camera transformation between the two that maximises photoconsistency. As described in the original publication we solve the transformation iteratively with a coarse-to-fine scheme in the form of a four level image pyramid.

1) *Preprocessing*: For both pairs we perform preprocessing on the RGB image and depth map. For each depth map we convert raw sensor values to a metric depth map $\mathbf{M} \in \mathbb{R}$. Additionally, according to the original implementation only gray values of the color image are used thus we define an intensity image $\mathbf{I} = (\mathbf{I}_R * 0.299 + \mathbf{I}_G * 0.587 + \mathbf{I}_B * 0.114)$ with $\mathbf{I} \in \mathbb{N}$. Following this a four level intensity and depth pyramid is constructed using a 5×5 Gaussian kernel for downsampling. Each of these steps is carried out on the GPU acting in parallel with one GPU thread per pixel. For the pair $[\mathbf{I}_{n+1}, \mathbf{M}_{n+1}]$ the partial derivatives $\frac{\partial \mathbf{I}_{n+1}}{\partial x}$ and $\frac{\partial \mathbf{I}_{n+1}}{\partial y}$ are also computed. A 3×3 Sobel operator is used for this computation, coupled with a 3×3 Gaussian blur with $\sigma = 0.8$. Again this step is carried out entirely on the GPU with one thread per pixel.

2) *Precomputation*: An optimisation introduced in the open source OpenCV CPU implementation only selects point

Algorithm 1: Interest Point Accumulation

Input: $\frac{\partial I_{n+1}}{\partial x}$ and $\frac{\partial I_{n+1}}{\partial y}$ intensity image derivatives
 s minimum gradient scale for pyramid level

Output: \mathcal{L} list of interest points
 $k_{\mathcal{L}}$ global point count

Data: α thread block x-dimension
 β thread block y-dimension
 γ pixels per thread
 ι shared memory local list
 κ shared memory local index
blockIdx CUDA block index
threadIdx CUDA thread index

in parallel do

```

i ←  $\beta * \mathbf{blockIdx.y} + \mathbf{threadIdx.y}$ 
j ←  $\alpha * \gamma * \mathbf{blockIdx.x} + \gamma * \mathbf{threadIdx.x}$ 
if  $\mathbf{threadIdx.x} = 0$  &&  $\mathbf{threadIdx.y} = 0$  then
   $\kappa \leftarrow 0$ 
  syncthreads();
  for  $l \leftarrow 0$  to  $\gamma$  do
     $\mathbf{p} \leftarrow (i, j + l)$ 
     $g^2 = \frac{\partial I_{n+1}}{\partial x}(\mathbf{p})^2 + \frac{\partial I_{n+1}}{\partial y}(\mathbf{p})^2$ 
    if  $g^2 \geq s$  then
       $idx \leftarrow \mathbf{atomicInc}(\kappa)$ 
       $\iota_{idx} \leftarrow \mathbf{p}$ 
  syncthreads();
   $b \leftarrow \alpha * \gamma * \mathbf{threadIdx.y} + \gamma * \mathbf{threadIdx.x}$ 
  for  $l \leftarrow 0$  to  $\gamma$  do
     $a \leftarrow b + l$ 
    if  $a < \kappa$  then
       $idx \leftarrow \mathbf{atomicInc}(k_{\mathcal{L}})$ 
       $\mathcal{L}_{idx} \leftarrow \iota_a$ 

```

correspondences with a minimum gradient in the intensity image. This is implemented as a binary image mask in the OpenCV version. Similarly we implement this optimisation but use a list of interest points over a mask. Compiling this list of points as a parallel operation is done using a basic parallel reduction exploiting shared memory in each CUDA thread block as inspired by a similar operation by van den Braak *et al.* [20]. Algorithm 1 lists the operation as it would operate for each level of the pyramid.

In the computation of the Jacobian matrix the projection of each point in \mathbf{M}_n is required. For each pyramid level the 3D projection $\mathbf{V}_n(\mathbf{p})$ of each point \mathbf{p} in the depth map is computed prior to beginning iteration with $\mathbf{V} \in \mathbb{R}^3$. Only projecting certain points based on a condition results in performance hindering branching and a reduction in pipelining. Empirically it was found to be faster to simply project the entire depth map rather than only project points required in correspondences. Given the intrinsic camera calibration matrix \mathbf{K} of the camera we can obtain the principal points c_x and c_y and the focal lengths f_x and f_y . The 3D reconstruction of each point \mathbf{p} is computed in parallel with one thread per point as $\mathbf{V}_n(\mathbf{p}) = \left(\frac{(\mathbf{p}_x - c_x)\mathbf{M}_n(\mathbf{p})}{f_x}, \frac{(\mathbf{p}_y - c_y)\mathbf{M}_n(\mathbf{p})}{f_y}, \mathbf{M}_n(\mathbf{p}) \right)^\top$

3) *Iterative Transformation Estimation:* Our iterative estimation process takes two main steps; (i) populating a list of valid correspondences from the precomputed list of interest points and (ii) solving the linear system for an incremental transformation and concatenating these transformations. The first step involves a reduction similar to the one in Algorithm

1, but rather than reducing from a 2D array to a 1D array it reduces from a 1D array to another 1D array; a distinction which results in a notable difference in implementation. On the first iteration for frame n we set the estimated camera transformation matrix \mathbf{T}_n to the identity, where

$$\mathbf{T}_n = \begin{bmatrix} \mathbf{R}_n & \mathbf{t}_n \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad (1)$$

with a 3×3 rotation matrix \mathbf{R}_n and a 3×1 translation vector \mathbf{t}_n . Before each iteration we compute the projection of \mathbf{T}_n into the image before uploading to the GPU as

$$\mathbf{R}_n^I = \mathbf{K}\mathbf{R}_n\mathbf{K}^{-1}, \quad \mathbf{t}_n^I = \mathbf{K}\mathbf{t}_n. \quad (2)$$

Algorithm 2 lists the process of populating a list of point correspondences from the list of interest points which can then be used to construct the Jacobian. With a list of valid correspondences we need only solve a least-squares equation

$$\arg \min_{\xi} \|\mathbf{J}_{rgbd}\xi + \mathbf{r}_{rgbd}\|^2 \quad (3)$$

to compute an improved camera transformation estimate

$$\mathbf{T}'_n = \exp(\hat{\xi})\mathbf{T}_n \quad (4)$$

$$\hat{\xi} = \begin{bmatrix} [\boldsymbol{\omega}]_{\times} & \mathbf{u} \\ 0 & 0 & 0 & 0 \end{bmatrix} \quad (5)$$

with $\xi = [\boldsymbol{\omega}^\top \mathbf{u}^\top]^\top$, $\boldsymbol{\omega} \in \mathbb{R}^3$ and $\mathbf{u} \in \mathbb{R}^3$. We first normalise the intensity difference sum σ computed in Algorithm 2 to enable a weighted optimisation $\sigma' = \sqrt{\sigma/k_c}$. Computation of the σ value in parallel is in fact an optimisation exploiting the atomic arithmetic functions available in the CUDA API. From here \mathbf{J}_{rgbd} and \mathbf{r}_{rgbd} can be populated according to the original algorithm documented by Steinbruecker *et al.* [8], including usage of σ' for weighting. Equation 3 is then solved using a tree reduction on the GPU followed by Cholesky factorization of the linear system on the CPU.

V. ODOMETRY ESTIMATION

In this section we describe the various combinations of odometry estimation algorithms we employed. We detail the combination of the FOVIS visual odometry system with both ICP-based and RGB-D-based pose estimators and the combination of KinectFusion’s original ICP estimator with Steinbruecker *et al.*’s RGB-D visual odometry system.

A. FOVIS Integration

The FOVIS visual odometry system of Huang *et al.* relies on FAST feature correspondences within the RGB frame [10]. Given that this is essentially using sparsely sampled features to compute a camera pose we opted for an integration strategy that allows our system to dynamically switch between FOVIS and any other estimator depending on some error metric. In particular, one problem which was quite evident was the way the ICP estimator would “slip” along corridors and other planar areas with a lack of 3D features. For this reason we chose a conservative switching strategy that favors the FOVIS estimator in the event of a disagreement in the estimate of the translation component.

Algorithm 2: Correspondence Accumulation

Input: \mathcal{L} list of interest points
 d_δ maximum change in point depth
 $[\mathbf{I}_n, \mathbf{M}_n]$ previous intensity depth pair
 $[\mathbf{I}_{n+1}, \mathbf{M}_{n+1}]$ current intensity depth pair
 \mathbf{R}_n^I camera rotation in image
 \mathbf{t}_n^I camera translation in image
Output: \mathcal{C} correspondence list of the form $(\mathbf{p}, \mathbf{p}', \Delta)$
 k_C global point count
 σ global intensity difference sum
Data: α thread block x-dimension
 γ pixels per thread
 ι shared memory local list
 κ shared memory local index
blockIdx CUDA block index
threadIdx CUDA thread index

in parallel do

```
 $i \leftarrow \alpha * \gamma * \text{blockIdx}.x + \gamma * \text{threadIdx}.x$ 
if  $\text{threadIdx}.x = 0$  then
   $\kappa \leftarrow 0$ 
syncthreads();
for  $l \leftarrow 0$  to  $\gamma$  do
   $\mathbf{p} \leftarrow \mathcal{L}_{i+l}$ 
   $z \leftarrow \mathbf{M}_{n+1}(\mathbf{p})$ 
  if isValid( $z$ ) then
     $(x', y', z')^\top \leftarrow z(\mathbf{R}_n^I(\mathbf{p}, 1)^\top) + \mathbf{t}_n^I$ 
     $\mathbf{p}' \leftarrow (\frac{x'}{z'}, \frac{y'}{z'})^\top$ 
    if isInImage( $\mathbf{p}'$ ) then
       $d \leftarrow \mathbf{M}_n(\mathbf{p}')$ 
      if isValid( $d$ ) &&  $|z' - d| \leq d_\delta$  then
         $idx \leftarrow \text{atomicInc}(\kappa)$ 
         $\iota_{idx} \leftarrow (\mathbf{p}, \mathbf{p}', \mathbf{I}_{n+1}(\mathbf{p}) - \mathbf{I}_n(\mathbf{p}'))$ 
  syncthreads();
   $b \leftarrow \gamma * \text{threadIdx}.x$ 
  for  $l \leftarrow 0$  to  $\gamma$  do
     $a \leftarrow b + l$ 
    if  $a < \kappa$  then
      atomicAdd( $\sigma, \iota_a^2$ )
       $idx \leftarrow \text{atomicInc}(k_C)$ 
       $\mathcal{C}_{idx} \leftarrow \iota_a$ 
```

This is based on the assumption that FOVIS odometry rarely “slips” as ICP does in planar environments (supported by consistent maximum frame-to-frame errors shown in Table II). The switching process for FOVIS and any other estimator (ICP, RGB-D, etc.) is as follows:

FOVIS Switching Strategy: From an estimator (e.g. ICP) we have an incremental camera transformation matrix \mathbf{T} . Where \mathbf{T}_F is the FOVIS camera transformation and \mathbf{T}_O is the estimator’s camera transformation, if $|\|\mathbf{t}_F\|_2 - \|\mathbf{t}_O\|_2| > \mu$ then use \mathbf{T}_F as the estimated incremental transformation, otherwise use \mathbf{T}_O . Emperically a value of $\mu = 0.03m$ was found to deliver adequate performance. The chosen transformation is then used to compute the next camera pose before the KinectFusion process continues on as normal. As shown in Section VII, in certain environments FOVIS alone can power a dense reconstruction when estimating odometry using only sparse features. Occasionally choosing the FOVIS estimate over the ICP estimate is unwise particularly in geometrically rich scenes, however relying on ICP too much increases the risk of a catastrophically large erroneous odometry estimate.

B. RGB-D and ICP Integration

To combine the color and depth information in the motion estimation we find the motion parameters ξ that minimize the combined sum of the RGB-D and ICP cost. The cost for ICP is the distance of each point in the current view to the corresponding point in the model

$$\mathbf{E}_{icp} = \sum_k \left\| \left(\mathbf{v}^k - \exp(\hat{\xi}) \mathbf{T} \mathbf{v}_n^k \right) \cdot \mathbf{n}^k \right\|^2, \quad (6)$$

where \mathbf{v}_n^k is the k -th vertex in frame n , $\mathbf{v}^k, \mathbf{n}^k$ are the corresponding vertex and normal respectively in the model, and \mathbf{T} is the current estimate of the transformation from current frame to the model frame. For simplicity of notation we omit conversions between 3-vectors (as needed for dot and cross products) and their corresponding homogeneous 4-vectors (as needed for multiplications with \mathbf{T}). If we linearize the transformation around the identity we get

$$\mathbf{E}_{icp} \approx \sum_k \left\| \left(\mathbf{v}^k - (\mathbf{I} + \hat{\xi}) \mathbf{T} \mathbf{v}_n^k \right) \cdot \mathbf{n}^k \right\|^2 \quad (7)$$

$$= \sum_k \left\| \left(\mathbf{v}^k - \mathbf{T} \mathbf{v}_n^k \right) \cdot \mathbf{n}^k - \hat{\xi} \mathbf{T} \mathbf{v}_n^k \cdot \mathbf{n}^k \right\|^2 \quad (8)$$

$$= \sum_k \left\| \begin{bmatrix} -\mathbf{T} \mathbf{v}_n^k \times \mathbf{n}^k \\ -\mathbf{n}^k \end{bmatrix}^\top \xi + (\mathbf{v}^k - \mathbf{v}_n^k) \cdot \mathbf{n}^k \right\|^2 \quad (9)$$

$$= \|\mathbf{J}_{icp} \xi + \mathbf{r}_{icp}\|^2 \quad (10)$$

and in a similar manner we get \mathbf{J}_{rgb} and \mathbf{r}_{rgb} for the color (intensity) correspondences, according to the original algorithm by Steinbruecker *et al.* [8].

The sum of the RGB-D and ICP cost is defined as

$$\mathbf{E} = \mathbf{E}_{icp} + w \cdot \mathbf{E}_{rgb}, \quad (11)$$

where w is the weight and was set empirically to 0.1 to reflect the difference in metrics used for ICP and RGB-D costs. For each step we minimize the linear least-squares problem by solving the normal equations

$$\begin{bmatrix} \mathbf{J}_{icp} \\ v \mathbf{J}_{rgb} \end{bmatrix}^\top \begin{bmatrix} \mathbf{J}_{icp} \\ v \mathbf{J}_{rgb} \end{bmatrix} \xi = \begin{bmatrix} \mathbf{J}_{icp} \\ v \mathbf{J}_{rgb} \end{bmatrix}^\top \begin{bmatrix} \mathbf{r}_{icp} \\ \mathbf{r}_{rgb} \end{bmatrix} \quad (12)$$

$$(\mathbf{J}_{icp}^\top \mathbf{J}_{icp} + w \mathbf{J}_{rgb}^\top \mathbf{J}_{rgb}) \xi = \mathbf{J}_{icp}^\top \mathbf{r}_{icp} + v \mathbf{J}_{rgb}^\top \mathbf{r}_{rgb}, \quad (13)$$

where $v = \sqrt{w}$. The products $\mathbf{J}^\top \mathbf{J}$ and $\mathbf{J}^\top \mathbf{r}$ are computed on the GPU using a tree reduction. The normal equations are then solved on the CPU using Cholesky factorization.

VI. COLOR FUSION

As discussed in Section II the keyframe approach to mapping color is the most popular. We extend an existing implementation for surface color information integration in our color estimation pipeline. The method performs a moving average in a similar fashion to the surface reconstruction. The result is a system for estimating color that updates surface color in tandem with the surface itself while averaging out noise, sensor artifacts, and other artifacts introduced by various optical phenomena.

A. Color Integration

We extend an existing color integration technique made available in the Point Cloud Library (PCL) developed by Anatoly Baksheev of Itseez [21]. The technique uses a separate color volume in GPU memory with the same dimensions as the TSDF volume. The color volume containing the fused set of frames from 1 to n is denoted as $\mathbf{C}_n(\mathbf{p})$ where $\mathbf{p} \in \mathbb{N}^3$ is the 3D coordinate of a voxel within the current volume coordinate frame. Each element of the color volume is made up of four 8-bit integers, packed into a single 32-bit integer

$$\mathbf{C}_n(\mathbf{p}) \mapsto [\mathbf{RGB}_n(\mathbf{p}), \mathbf{W}_n(\mathbf{p})], \quad (14)$$

where $\mathbf{RGB}_n(\mathbf{p}) \in \mathbb{N}^3$ stores the red, green and blue components of the element at \mathbf{p} and $\mathbf{W}_n(\mathbf{p})$ stores the weight. The implementation of this volume including the update scheme is available in the PCL open source implementation of KinectFusion known as KinFu [22]. In our system we have extended the method and coupled it more tightly to the surface reconstruction process. At the time of writing the KinFu implementation does not provide live real-time surface coloring or attempt to avoid the integration of unreliable color measurements as we do.

When extracting the surface from the TSDF for live rendering color extraction is made easy by a one-to-one mapping maintained between the surface volume and the color volume. After detection of a zero crossing according to the original KinectFusion raycast at a point \mathbf{p} [9], the color for \mathbf{p} on the surface can be found at $\mathbf{C}_n(\mathbf{p})$. Additionally we trilinearly interpolate the value around this point to alleviate some of the visual artifacts introduced to the render by the discrete voxel structure of the TSDF.

B. Artifact Reduction

The surface coloring is often inaccurate around the edges of objects. This can be caused by inaccuracy in the calibration between the depth and color cameras, errors in depth measurements or light diffraction around objects. Typically there are obvious depth discontinuities around such edges which can cause the background to blend with the model. This problem is addressed by not updating points on the surface close to depth discontinuities and using the angle between the surface and the camera to weight the color update. A point is determined to be on a boundary if some of its neighbors are more than a given distance away from it, considering a neighborhood of 7×7 points. When each frame is prepared a normal vector is computed for each point in the depth image. The angle θ can be computed by

$$\cos(\theta) = [0 \ 0 \ 1] \cdot \mathbf{n} = n_z \quad (15)$$

and used to weight each color update in the color volume, resulting in colors viewed “straight on” being weighted higher than those viewed at an angle.

C. Coloring Results

Figure 3 shows a comparison between a model colored without artifact reduction and a model colored with the two enhancements listed in Section VI-B. It can be clearly



Fig. 3. Shown on the left is the model using plain coloring and on the right is the same model colored using angle of observation weighting and the discontinuity check described in Section VI-B.

seen in Figure 3 that the enhancements we have integrated greatly reduce the amount of bloom and color bleed around the edges of the model. Additional results can be seen in our video submission available at <http://www.youtube.com/watch?v=MEugh12dcYA>.

There is a small computational penalty for these enhancements. An increase in execution time of about 2ms per frame was measured. Additionally, the usage of a separate color volume requires twice as much memory as is needed for an uncolored model. This increases the GPU memory requirement of the system to 1GB, which in reality facilitates the need for a 2GB GPU as there is also GPU memory required for the operating system and visualisation components.

VII. RESULTS

We evaluate our system both quantitatively and qualitatively. We present results on frame-to-frame tracking and overall trajectory accuracy on selected data from the Freiburg ground truth dataset [23]. Additionally we present computational performance results. We also present qualitative results in the associated submission video and figures.

A. Quantitative Results

Table I shows the absolute trajectory error for each algorithm on each dataset. Tables II and III show the relative frame-to-frame error in translation and rotation respectively. From these results alone the algorithms evaluated appear to have comparable performance with the exception of a small set of outlier results. Thus we have derived a ranking score that aims to rank the algorithms with the most consistent performance higher than those with more sporadic performance.

1) *Relative Ranking Algorithm:* Given a set of m algorithms \mathcal{A} and a set of n datasets \mathcal{D} we denote the error achieved by a given algorithm \mathcal{A}_i on a dataset \mathcal{D}_j as $\mathcal{E}(\mathcal{A}_i, \mathcal{D}_j)$ (this could be for example maximum translation error or median rotational error, etc.). The score \mathcal{S} for an algorithm \mathcal{A}_i given \mathcal{A} and \mathcal{D} is defined as

$$\mathcal{S}(\mathcal{A}_i, \mathcal{A}, \mathcal{D}) = \sum_j^n \frac{\mathcal{E}(\mathcal{A}_i, \mathcal{D}_j)}{n \sum_k^m \mathcal{E}(\mathcal{A}_k, \mathcal{D}_j)}. \quad (16)$$

This ranking encapsulates the relative error each algorithm produces compared to each other. These results show that in each of the median scores ICP+RGB-D is ranked first in consistency and FOVIS/ICP+RGB-D is ranked second. For maximum error consistency in each test either ICP+RGB-D

TABLE IV

RELATIVE ROOT MEAN SQUARE ERROR (RMSE) OF DRIFT IN METERS
PER SECOND FOR GROUND TRUTH DATA.

Dataset	fr1/desk	fr2/desk	fr1/room	fr2/large_no_loop
ICP	0.1198m	0.0453m	0.1187m	1.0130m
RGB-D	0.0558m	0.0321m	0.1028m	0.2254m
FOVIS	0.0604m	0.0136m	<i>0.0642m</i>	0.0960m
FOVIS/ICP	0.1108m	0.0459m	0.0776m	0.2329m
FOVIS/RGB-D	0.0555m	0.0342m	0.0708m	<i>0.1412m</i>
ICP+RGB-D	0.0393m	0.0208m	0.0622m	0.1795m
FOVIS/ICP+RGB-D	<i>0.0395m</i>	<i>0.0207m</i>	0.0755m	0.1652m



Fig. 6. Shown from top to bottom are orthographic projections of the meshes produced using FOVIS, ICP, RGB-D, and finally ICP+RGB-D for camera tracking. Color has been removed for the models for clarity. Each model was aligned according to the initial camera position.

or FOVIS/ICP+RGB-D is always ranked second, while either FOVIS or FOVIS/RGB-D is always ranked first. The score rankings are visualised in Figures 4 and 5. These results are consistent with results observed in the qualitative analysis.

In Table IV we also provide a set of results for each algorithm using the root mean square error (RMSE) of the drift in meters per second metric, which has recently become a more favorable metric where error magnitude between frames is close to the noise level of the motion capture system [23]. In Tables I through IV best quantities are marked in bold and second best in italics.

B. Qualitative Results

Figures 6 and 7 show qualitative results for two datasets. The first dataset, shown in Figure 6, shows the performance of FOVIS, ICP, RGB-D and ICP+RGB-D tracking on a dataset where a human carried a camera at torso level down a corridor and a small set of steps with highly varied lighting. At some points in this dataset the RGB image is pure black due to very low overhead lighting. As expected, both the FOVIS and RGB-D tracking suffer from the lack of visual features at some points, while the ICP tracking fails in areas where there are no obvious 3D features like door frames and radiators. The ICP+RGB-D tracking is strong and robust throughout, showing no signs of failure anywhere along the trajectory. Additionally ICP+RGB-D shows significantly less signs of drift than the other tracking algorithms.

Figure 7 shows the second dataset. This dataset was again captured by a human operating a camera at torso level. The

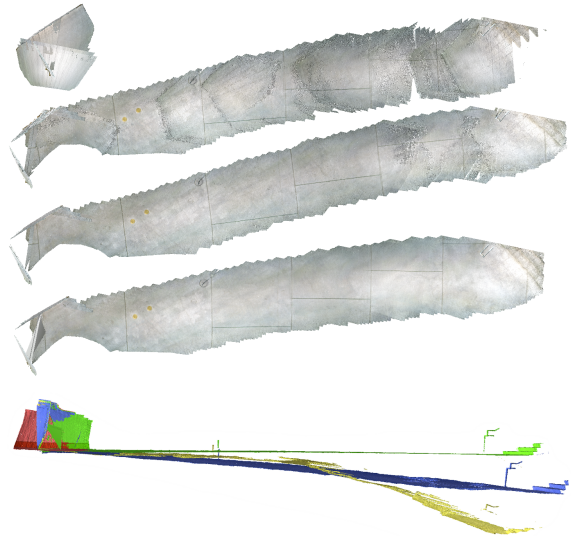


Fig. 7. Shown from top to bottom are orthographic projections of the meshes produced using ICP, RGB-D, FOVIS and finally ICP+RGB-D for camera tracking. The fifth element shown at the bottom of the figure is an elevation side view of all four models aligned according to the initial camera position with ICP shown in red, RGB-D shown in yellow, FOVIS shown in blue and ICP+RGB-D shown in green.

TABLE V

COMPUTATIONAL PERFORMANCE OF EACH ALGORITHM ON THE
FR1/ROOM DATASET.

Timing	Algorithm (ms)			
	Avg	Min	Max	StdDev
ICP	8.780	8.450	9.210	0.077
RGB-D	10.68	5.820	15.81	1.818
FOVIS	16.18	12.66	20.80	1.584
FOVIS/ICP	15.66	7.300	27.52	3.866
FOVIS/RGB-D	16.50	15.14	19.62	0.442
ICP+RGB-D	18.48	12.79	23.52	1.790
FOVIS/ICP+RGB-D	23.81	19.65	27.77	1.354

trajectory begins facing the corner of a building, rotating around and moving forward across a large room until reaching the bottom of a staircase. The most notable feature of this dataset is the fact that only the floor is visible for almost all of the trajectory, which contains only a moderate amount of visual features. As expected, ICP fails as soon as the floor is reached. RGB-D performs slightly better but produces a very bad elevation estimate. FOVIS performs much better but again produces a bad elevation estimate. The ICP+RGB-D tracker performs best out of all four algorithms with a very strong trajectory estimate in all dimensions, evident by the floor pattern texture and side on elevation views in Figure 7. A number of additional qualitative results are presented in our video submission available at <http://www.youtube.com/watch?v=MEugh12dcYA>, showing independent ICP, RGB-D and FOVIS odometry estimators versus the combined ICP+RGB-D estimator.

C. Computational Performance

Tables V and VI show execution time statistics for each algorithm on the fr1/room dataset. The test platform used was a standard desktop PC running Ubuntu 12.04 with an Intel Core i7-3960X CPU at 3.30GHz, 16GB of RAM and

TABLE I

ABSOLUTE TRAJECTORY ERROR FOR GROUND TRUTH DATA SHOWING MEDIAN AND MAXIMUM TRANSLATIONAL ERRORS.

Dataset	fr1/desk		fr2/desk		fr1/room		fr2/large_no_loop		Rank Score	
	Median	Max	Median	Max	Median	Max	Median	Max	Median	Max
ICP	0.028m	0.396m	0.087m	0.456m	0.326m	0.719m	3.298m	5.639m	0.2143	0.2409
RGB-D	0.094m	0.138m	0.132m	0.372m	0.447m	0.872m	0.261m	0.466m	0.1423	0.1120
FOVIS	0.221m	0.799m	0.112m	0.217m	0.238m	0.508m	0.273m	0.897m	0.1443	0.1383
FOVIS/ICP	0.284m	1.062m	0.095m	0.463m	0.193m	0.547m	1.225m	2.097m	0.1893	0.2121
FOVIS/RGB-D	0.094m	0.137m	0.121m	0.318m	0.279m	0.551m	0.582m	0.904m	0.1283	0.0959
ICP+RGB-D	0.069m	0.234m	0.119m	0.362m	0.158m	0.421m	0.256m	0.878m	0.0904	<i>0.0997</i>
FOVIS/ICP+RGB-D	0.068m	0.231m	0.118m	0.346m	0.152m	0.419m	0.309m	1.032m	<i>0.0911</i>	0.1010

TABLE II

RELATIVE FRAME-TO-FRAME ERROR FOR GROUND TRUTH DATA SHOWING MEDIAN AND MAXIMUM TRANSLATIONAL ERRORS.

Dataset	fr1/desk		fr2/desk		fr1/room		fr2/large_no_loop		Rank Score	
	Median	Max	Median	Max	Median	Max	Median	Max	Median	Max
ICP	0.0043m	0.3039m	0.0019m	0.0681m	0.0037m	0.2178m	0.0372m	0.2912m	0.1592	0.3160
RGB-D	0.0066m	0.0296m	0.0043m	0.0162m	0.0054m	0.2261m	0.0174m	0.2669m	0.1739	0.1557
FOVIS	0.0059m	0.0419m	0.0024m	0.0122m	0.0051m	0.0763m	0.0112m	0.1056m	0.1300	0.0752
FOVIS/ICP	0.0058m	0.2915m	0.0019m	0.0368m	0.0041m	0.0762m	0.0154m	0.2043m	0.1245	0.2067
FOVIS/RGB-D	0.0065m	0.0297m	0.0043m	0.0187m	0.0053m	0.0762m	0.0164m	0.1078m	0.1703	0.0813
ICP+RGB-D	0.0056m	0.0655m	0.0025m	0.0108m	0.0045m	0.0892m	0.0087m	0.1101m	0.1194	0.0851
FOVIS/ICP+RGB-D	0.0057m	0.0652m	0.0026m	0.0107m	0.0045m	0.0763m	0.0094m	0.1056m	<i>0.1228</i>	<i>0.0800</i>

TABLE III

RELATIVE FRAME-TO-FRAME ERROR FOR GROUND TRUTH DATA SHOWING MEDIAN AND MAXIMUM ROTATIONAL ERRORS.

Dataset	fr1/desk		fr2/desk		fr1/room		fr2/large_no_loop		Rank Score	
	Median	Max	Median	Max	Median	Max	Median	Max	Median	Max
ICP	0.0103°	9.9997°	0.0039°	2.6484°	0.0066°	16.4409°	0.0079°	7.3459°	0.1828	0.3094
RGB-D	0.0077°	1.8041°	0.0037°	1.3458°	0.0055°	8.4932°	0.0051°	3.4535°	0.1425	0.1300
FOVIS	0.0071°	8.0873°	0.0032°	1.2952°	0.0053°	1.9648°	0.0043°	1.4295°	0.1278	0.1187
FOVIS/ICP	0.0092°	5.3246°	0.0039°	1.2601°	0.0066°	9.6321°	0.0051°	5.7109°	0.1581	0.1853
FOVIS/RGB-D	0.0076°	1.8010°	0.0038°	1.3778°	0.0055°	2.7109°	0.0051°	1.5468°	0.1430	0.0781
ICP+RGB-D	0.0070°	2.8915°	0.0032°	1.3137°	0.0050°	3.0983°	0.0039°	2.4962°	0.1227	0.0972
FOVIS/ICP+RGB-D	0.0070°	2.8658°	0.0031°	1.2775°	0.0051°	2.4799°	0.0040°	1.4224°	<i>0.1230</i>	<i>0.0812</i>

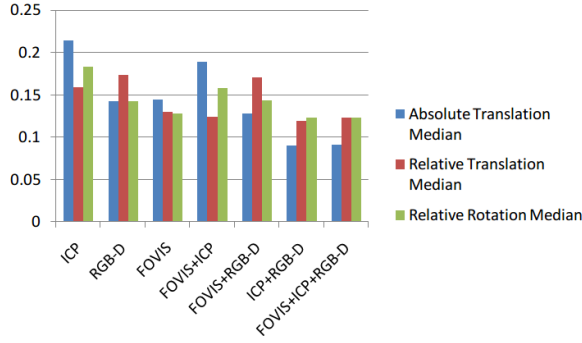


Fig. 4. Ranking score for each algorithm's median error.

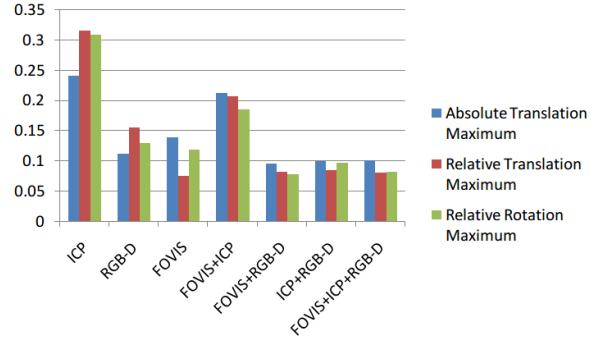


Fig. 5. Ranking score for each algorithm's maximum error.

TABLE VI

COMPUTATIONAL PERFORMANCE OF THE FULL KINTINUOUS PIPELINE ON THE FR1/ROOM DATASET.

Timing	Full Pipeline (ms)			
	Avg	Min	Max	StdDev
ICP	23.93	13.38	33.55	2.463
RGB-D	25.20	19.82	36.56	2.412
FOVIS	30.80	25.83	39.36	2.305
FOVIS/ICP	25.70	16.19	42.75	4.699
FOVIS/RGB-D	26.04	21.98	34.15	2.016
ICP+RGB-D	28.44	23.10	37.73	2.481
FOVIS/ICP+RGB-D	34.62	29.18	45.56	2.628

an nVidia GeForce 680GTX GPU with 2GB of memory. The results in Table V show that each algorithm is more than capable of operating at the sensor frame rate of 30Hz,

with the maximum execution times well below 33ms. From a robotics and control stand point the results in Table VI are more interesting. On average all algorithms except FOVIS/ICP+RGB-D are capable of functioning in the full mapping pipeline at the sensor frame rate of 30Hz. In our previous work we demonstrated that a capture rate of 15Hz is sufficient to produce high quality dense meshes [1]. Using 66ms as the target execution time our pipeline is capable of functioning at 15Hz with at least 20ms spare execution time each frame. Such a low latency system is crucial for real-time closed loop controlled robotic platforms.



Fig. 8. Sample models shown in our video submission available at <http://www.youtube.com/watch?v=MEugh12dcYA>. Left: dinosaur model from the Australian Museum, Sydney. Middle: PR2 lab on the fourth floor of the MIT Stata Center. Right: hotel room.

VIII. CONCLUSIONS

This paper has presented a system for improved RGB-D camera pose tracking that yields high quality color surface models with low visual artifacts. Our system is based on a novel GPU implementation of an existing RGB-D visual odometry algorithm, which is evaluated against and in combination with ICP and FOVIS. Our results demonstrate the ability of the Kintinuuous system to produce high quality dense color maps with robust tracking in challenging environments while still executing in low latency real-time. While the system can be made to fail in extreme conditions, such as very high camera velocity or a lack of both visual and depth features, in general the system has performed well in extensive testing in a variety of indoor scenes. In ongoing work, we are addressing: (1) updating areas of the map we are returning to that have exited the TSDF volume; and (2) closing large loops via SLAM pose-graph optimisation.

ACKNOWLEDGMENT

Research presented in this paper was funded by a Strategic Research Cluster grant (07/SRC/I1168) by Science Foundation Ireland under the Irish National Development Plan, the Embark Initiative of the Irish Research Council and by ONR grants N00014-10-1-0936, N00014-11-1-0688, and N00014-12-10020. The authors would also like to thank Guillaume Gales and Maurice Fallon.

REFERENCES

- [1] T. Whelan, J. McDonald, M. Kaess, M. Fallon, H. Johannsson, and J. Leonard, "Kintinuuous: Spatially Extended KinectFusion," in *3rd RSS Workshop on RGB-D: Advanced Reasoning with Depth Cameras*, (Sydney, Australia), July 2012.
- [2] F. Endres, J. Hess, N. Engelhard, J. Sturm, D. Cremers, and W. Burgard, "An evaluation of the RGB-D SLAM system," in *Proc. of the IEEE Int. Conf. on Robotics and Automation (ICRA)*, (St. Paul, MA, USA), May 2012.
- [3] P. Henry, M. Krainin, E. Herbst, X. Ren, and D. Fox, "RGB-D mapping: Using Kinect-style depth cameras for dense 3D modeling of indoor environments," *The Int. Journal of Robotics Research*, 2012.
- [4] C. Audras, A. I. Comport, M. Meilland, and P. Rives, "Real-time dense RGB-D localisation and mapping," in *Australian Conf. on Robotics and Automation*, (Monash University, Australia), December 2011.
- [5] R. A. Newcombe, S. J. Lovegrove, and A. J. Davison, "DTAM: Dense tracking and mapping in real-time," in *Computer Vision (ICCV), 2011 IEEE Int. Conf. on*, pp. 2320–2327, November 2011.
- [6] R. A. Newcombe, S. Izadi, O. Hilliges, D. Molyneaux, D. Kim, A. J. Davison, P. Kohli, J. Shotton, S. Hodges, and A. Fitzgibbon, "KinectFusion: Real-time Dense Surface Mapping and Tracking," in *Proc. of the 2011 10th IEEE Int. Symposium on Mixed and Augmented Reality*, ISMAR '11, (Washington, DC, USA), pp. 127–136, 2011.
- [7] J. Stuehmer, S. Gumhold, and D. Cremers, "Real-time dense geometry from a handheld camera," in *Pattern Recognition (Proc. DAGM)*, (Darmstadt, Germany), pp. 11–20, September 2010.
- [8] F. Steinbruecker, J. Sturm, and D. Cremers, "Real-Time Visual Odometry from Dense RGB-D Images," in *Workshop on Live Dense Reconstruction with Moving Cameras at the Int. Conf. on Computer Vision (ICCV)*, November 2011.
- [9] S. Izadi, D. Kim, O. Hilliges, D. Molyneaux, R. Newcombe, P. Kohli, J. Shotton, S. Hodges, D. Freeman, A. Davison, and A. Fitzgibbon, "KinectFusion: Real-Time 3D reconstruction and interaction using a moving depth camera," in *Proc. of the 24th annual ACM symposium on User interface software and technology*, UIST '11, (New York, NY, USA), pp. 559–568, ACM, 2011.
- [10] A. S. Huang, A. Bachrach, P. Henry, M. Krainin, D. Maturana, D. Fox, and N. Roy, "Visual odometry and mapping for autonomous flight using an RGB-D camera," in *Int. Symposium on Robotics Research (ISRR)*, (Flagstaff, Arizona, USA), August 2011.
- [11] K. Pirker, M. R  ther, G. Schweighofer, and H. Bischof, "GPSlam: Marrying sparse geometric and dense probabilistic visual mapping," in *Proc. of the British Machine Vision Conf.*, pp. 115.1–115.12, 2011.
- [12] "ReconstructMe FAQ," <http://reconstructme.net/usage/#multiscan>, August 10th 2012.
- [13] "KinectFusion extensions to large scale environments," <http://www.pointclouds.org/blog/srcs/fheredia/index.php>, August 10th 2012.
- [14] H. Roth and M. Vona, "Moving volume KinectFusion," in *British Machine Vision Conf. (BMVC)*, (Surrey, UK), September 2012.
- [15] "OpenCV Meeting Notes," <http://pr.willowgarage.com/wiki/OpenCVMeetingNotes/Minutes%202012-03-05>, August 10th 2012.
- [16] P. Henry, M. Krainin, E. Herbst, X. Ren, and D. Fox, "RGB-D mapping: Using depth cameras for dense 3D modeling of indoor environments," in *Proc. of Int. Symposium on Experimental Robotics (ISER)*, In RGB-D: Advanced Reasoning with Depth Cameras Workshop in conjunction with RSS, 2010.
- [17] J. St  ckler and S. Behnke, "Integrating depth and color cues for dense multi-resolution scene mapping using RGB-D Cameras," in *Proc. of the IEEE Int. Conf. on Multisensor Fusion and Information Integration (MFI)*, (Hamburg, Germany), September 2012.
- [18] B. Curless and M. Levoy, "A volumetric method for building complex models from range images," in *Proc. of the 23rd annual Conf. on Computer graphics and interactive techniques - SIGGRAPH '96*, (New York, New York, USA), pp. 303–312, August 1996.
- [19] Z. C. Marton, R. B. Rusu, and M. Beetz, "On fast surface reconstruction methods for large and noisy datasets," in *Proc. of the IEEE Int. Conf. on Robotics and Automation (ICRA)*, (Kobe, Japan), May 2009.
- [20] G.-J. van den Braak, C. Nugteren, B. Mesman, and H. Corporaal, "Fast hough transform on GPUs: Exploration of algorithm trade-offs," in *Advances Concepts for Intelligent Vision Systems*, vol. 6915 of *Lecture Notes in Computer Science*, pp. 611–622, 2011.
- [21] "Itseez," <http://www.itseez.com>, August 10th 2012.
- [22] R. B. Rusu and S. Cousins, "3D is here: Point cloud library (PCL)," in *IEEE Int. Conf. on Robotics and Automation (ICRA)*, (Shanghai, China), May 2011.
- [23] 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 Int. Conf. on Intelligent Robot Systems (IROS)*, October 2012.