2013 IEEE International Conference on Robotics and Automation (ICRA)
Karlsruhe, Germany, May 6-10, 2013

Fast Visual Odometry and Mapping from RGB-D Data

Ivan Dryanovski*, Roberto G. Valenti and Jizhong Xiao, *Senior Member, IEEE*

***Abstract*— An RGB-D camera is a sensor which outputs color and depth and information about the scene it observes. In this paper, we present a real-time visual odometry and mapping system for RGB-D cameras. The system runs at frequencies of 30Hz and higher in a single thread on a desktop CPU with no GPU acceleration required. We recover the unconstrained 6-DoF trajectory of a moving camera by aligning sparse features observed in the current RGB-D image against a model of previous features. The model is persistent and dynamically updated from new observations using a Kalman Filter. We formulate a novel uncertainty measure for sparse RGD-B features based on a Gaussian mixture model for the filtering stage. Our registration algorithm is capable of closing small-scale loops in indoor environments online without any additional SLAM back-end techniques.**

I. INTRODUCTION

An RGB-D camera is a device which provides two concurrent image streams: a conventional color image, and a depth image, containing a measure of the distance from the camera to each observed pixel. The depth information is recovered using a combination of an infrared projector and infrared camera built into the device. In recent years, low-cost RGB-D cameras such as the Microsoft Kinect have become widely popular in the robotics and computer vision research communities.

We examine the problem of recovering the unconstrained trajectory of an RGB-D camera, observing a static scene, from a sequence of time-consecutive RGB-D images $I_{0:n}$. The trajectory of the camera is represented as a sequence of poses $C_{0:n}$, each with 6 degrees of freedom (position and orientation). The estimated trajectory can be used as input to a control loop, or to build a global 3D map using the RGB-D data.

A common solution to this problem is to compute the pose of the camera incrementally, by aligning each image to the previous one. To do this, one would need to calculate the rigid transformation T_t which best aligns I_t to I_{t-1}. Each camera pose can thus be expressed as

$$C_t = C_{t-1} T_t \tag{1}$$

where the initial camera position is assumed to be the Identity transform ($C_0 = I$).

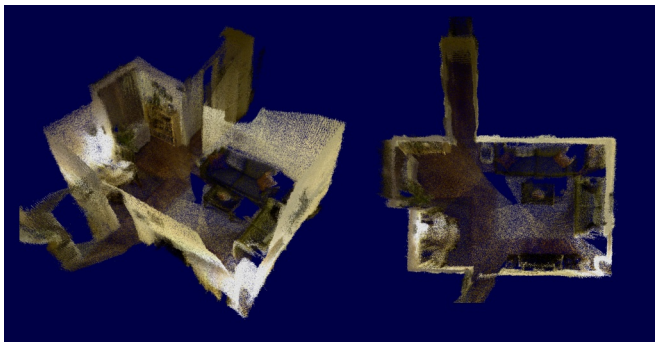

Fig. 1. 3D map of a room reconstructed with our method by going around the room with a handheld RGB-D camera. The loop is closed seamlessly by the algorithm. Left: side view. Right: orthographic top view.

In robotics, this is often referred to as *visual odometry* (VO), a term introduced by Nister et al. [8]. In their paper, they demonstrated a real-time system for a ground robot equipped with a stereo camera. The name is applied to any algorithm for trajectory estimation using input solely from a single or multiple cameras. An overview of the history of visual odometry, as well as state-of-the-art systems, is given by Scaramuzza and Fraundorfer [10].

The choice to perform the alignment only between the last pair of images (as opposed to the entire sequence) reduces the computational complexity of the algorithm to constant time with respect to the path length. This allows visual odometry systems to run in real time, at high update frequencies. The drawback to this design is that small errors in the incremental pose computation build up over time and introduce drift in the trajectory. A common approach to alleviate this is to perform the registration against the last m images. These techniques, termed *sliding window*, retain the constant time complexity of the pairwise registration. They can reduce the drift error considerably, but do not resolve the problem completely.

The challenge of obtaining a globally-correct trajectory over long distances, including revisiting locations, is known as *loop closing*. In their tutorial, Scaramuzza and Fraundorfer [10] classify these algorithms as V-SLAM, or Vision-based Simultaneous Localization And mapping. V-SLAM systems often rely on visual odometry as a building block.

This work is supported in part by U.S. Army Research Office under grant No. W911NF-09-1-0565, U.S. National Science Foundation under grants No. IIS- 0644127 and No. CBET-1160046, Federal High-Way Administration (FHWA) under grant No. DTFH61-12-H-00002 and PSC-CUNY under grant No. 65789-00-43.

*Ivan Dryanovski is with the Dept. of Computer Science, The Graduate Center, The City University of New York (CUNY), 365 Fifth Avenue, New York, NY 10016 (corresponding author, e-mail: idryanovski@gc.cuny.edu)

Roberto G. Valenti is with the Electrical Engineering Department, City College of New York, Convent Ave & 140th Street, New York, NY 10031 (e-mail: rvalent00@ccny.cuny.edu)

Jizhong Xiao is with the Electrical Engineering Department, City College of New York, Convent Ave & 140th Street, New York, NY 10031 (e-mail: jxiao@ccny.cuny.edu)

A. Related Work

Steinbrucker et al. [13] presented a system for frame-to-frame trajectory estimation by minimizing an energy function in the space of the dense depth data. Dryanovski et al. [2] presented a system for frame-to-frame registration using edge features, which used a high-frequency loop for sparse data and a low-frequency loop for dense data.

Henry et al. [4] presented a system which uses GPU-accelerated SIFT features. Images are aligned on a frame-to-frame basis, by using both sparse and dense data. Global refinement is performed offline using Sparse Bundle Adjustment.

Endres et al. [3] presented a system which uses sparse SURF, ORB, or GPU-accelerated SIFT descriptors (the choice is configurable). Images are aligned against a subset of previous frames of constant size, in order to increase robustness. The implementation requires multiple threads. Additional refinement of the trajectory can be performed offline.

Recently, Newcombe et al. [7] presented a system for RGB-D pose tracking and mapping. Their method aligns dense depth data against a model of a surface. The model is augmented with new data. The system requires a GPU-equipped computer.

B. Contribution

In this paper, we present a system for visual odometry which does not rely on frame-to-frame or sliding window techniques. We begin by computing the locations of sparse features in the incoming RGB-D image, and their corresponding 3D coordinates in the camera frame. Next, we align these features against a global model dataset of 3D features, expressed in the fixed coordinate frame. Aligning is performed with the ICP algorithm [14]. After calculating the transformation, the model is augmented with the new data. We associate features from the RGB-D image with features in the model, and update them using a Kalman Filter framework. Any features from the image which cannot be associated are inserted as new landmarks in the model set.

The model (which starts out empty) gradually grows in size as new data is accumulated. To guarantee constant-time performance, we place an upper bound on the number of points the model can contain. Once the model grows beyond that size, the oldest features are dropped to free up space for new ones.

To be able to perform the data association and filtering accurately, we develop a novel method for estimating the uncertainty in the depth reading of the RGB-D camera. The method is based on a Gaussian mixture model, and is able to accurately capture the high uncertainty of the depth in problematic areas such as object edges.

By performing alignment against a persistent model instead of only the last frame, we are able to achieve significant decrease in the drift of the pose estimation. We show that even with a small model size, we can accurately map out an environment the size of a room or office, and accurately close the loop without the need of any additional back-end processing techniques typically associated with V-SLAM algorithms.

We perform the trajectory estimation in real time, at rates of 30Hz or higher (the camera outputs the images at 30Hz, but our algorithm is able to process them faster). It uses a single thread, and does not require a GPU. Our implementation is publicly available under a free, open-source license.

II. UNCERTAINTY MODEL

In this section, we describe how to estimate the 3D coordinates and uncertainty of sparse features detected from the RGB-D image. Given a pixel $\mathbf{q} = [u, v, d]^T$, where u and v are the image coordinates, and d is the raw depth measurement of the RGB-D camera, we can express it as a 3D point $\mathbf{p} = [x, y, z]^T$, in the camera coordinate frame [5]:

$$z = \frac{Z_0}{1 + \frac{Z_0}{fb} d} \tag{2a}$$

$$x = \frac{z}{f}(u - c_x) \tag{2b}$$

$$y = \frac{z}{f}(v - c_y) \tag{2c}$$

We have the following parameters:

- Z_0: distance to the reference plane used for stereo matching (an internal device parameter)
- b: the baseline between the IR projector and camera
- f: focal distance of the IR camera
- c_x, c_y: IR image optical center

To obtain the uncertainty in z, Khoshelham and Elberink [5] treat d as a random variable with standard deviation σ_d. From equation 2a they derive

$$\sigma_z = \frac{1}{fb} \sigma_d \mu_z^2 \tag{3}$$

The expression with the calibrated parameters which the authors obtain is

$$\sigma_z = 1.45 \times 10^{-3} \mu_z^2 \tag{4}$$

where z is expressed in meters. The authors assume that the depth error in each pixel is independent of the pixels around it. Furthermore, they assume there is no uncertainty in the u and v readings.

A. Depth uncertainty through a Gaussian mixture model

We extend the model in two ways. First, the locations of features detected by a sparse feature detector are subject to error; therefore, we allow for uncertainty in the u and v dimensions. Second, we assume that the depth uncertainty is dependent not only on the depth readings of a given pixel, but also on its neighbors in a local window. We will show that using these assumptions, we can predict the magnitude of the depth uncertainty better than the previously published model.

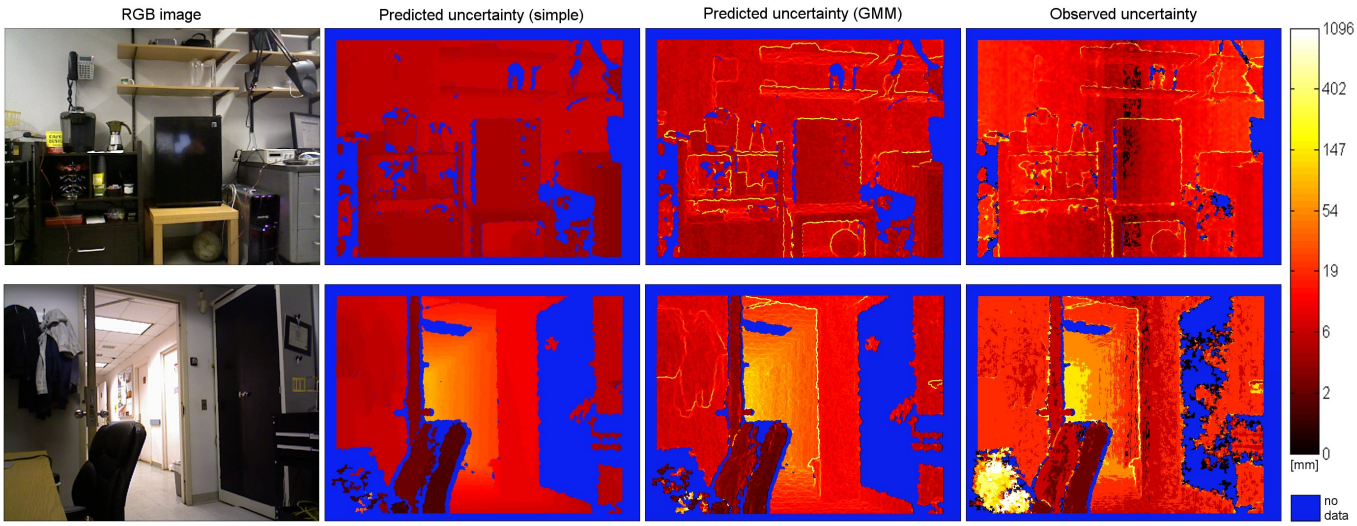

Fig. 2. Uncertainty analysis for the depth readings of an RGB-D camera. Left: RGB image, shown for visualization only. Right: *observed* (ground truth) uncertainty ($\bar{\sigma}_z$), obtained by taking 200 depth images of a static scene and calculating the mean and standard deviation on a per-pixel basis. Color is scaled as the log of the uncertainty. Center-left: uncertainty predicted from a single depth image, according to the simple model (σ_z). Center-right: uncertainty predicted from the same depth image, according to the Gaussian mixture model ($\widehat{\sigma}_z$). We demonstrate the Gaussian model uncertainty predicts the true uncertainty more accurately, especially around object edges.

Let us begin by assuming that the u and v are independent random variables distributed according to a normal distribution $N(\mu_u, \sigma_u)$ and $N(\mu_v, \sigma_v)$ respectively. Further, let σ_u and σ_v inform the following approximate Gaussian kernel of size 3×3:

$$W = \frac{1}{16}\begin{bmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{bmatrix} \tag{5}$$

Assuming that z is normally distributed, we can define a random variable $\widehat{z}$, which is a mixture of the z variables in a local window $\{i :\in [\mu_u - 1, \mu_u + 1], j \in [\mu_v - 1, \mu_v + 1]\}$. The weights of the mixture w_{ij} are chosen according to the kernel W. The mean and variance of the resulting Gaussian mixture are

$$\widehat{\mu}_z = \sum_{i,j} w_{ij} \left(\mu_{z_{ij}}\right) \tag{6a}$$

$$\widehat{\sigma}_z^2 = \sum_{i,j} w_{ij} \left(\sigma_{z_{ij}}^2 + \mu_{z_{ij}}^2\right) - \widehat{\mu}_z^2 \tag{6b}$$

At this stage, we have two alternative models for the uncertainty: σ_z, estimated according to the simple model in equation 3, and $\widehat{\sigma}_z$, estimated according to the Gaussian mixture model. To evaluate which model has more predictive power, we gather n depth images of a static scene. For each pixel in the image, we calculate the uncertainty $\bar{\sigma}_z$ in the metric depth z according to

$$\bar{\mu}_z = \frac{1}{n}\sum_{k=1}^{n} z_k \tag{7a}$$

$$\bar{\sigma}_z^2 = \frac{1}{n-1}\sum_{k=1}^{n} \left(\bar{\mu}_z - z_k\right)^2 \tag{7b}$$

We call this the *observed* uncertainty. When n is large (we used 200 images) we can assume that the observed uncertainty approaches the true uncertainty for the RGB-D camera measurement. Next, we take a single depth image, and generate the two *predicted* uncertainties σ_z and $\widehat{\sigma}_z$. Fig. 2 shows a comparison between the observed and predicted uncertainties according to both models. We note that the Gaussian mixture model predicts the uncertainty much better than the simple model, especially around the edges of objects. This comes from the fact that the Kinect camera produces readings for edge pixels which tend to jump from foreground to background.

B. 3D uncertainty

We can estimate the 3D uncertainty Σ of a point $\mathbf{p}$ from equations 2.

$$\Sigma = \begin{bmatrix} \sigma_x^2 & \sigma_{xy} & \sigma_{xz} \\ \sigma_{xy} & \sigma_y^2 & \sigma_{yz} \\ \sigma_{xz} & \sigma_{zy} & \sigma_z^2 \end{bmatrix} \tag{8}$$

where

$$\begin{aligned}
\sigma_x^2 &= \frac{\sigma_z^2(\mu_u - c_x)(\mu_v - c_y) + \sigma_u^2(\mu_z^2 + \sigma_z^2)}{f_x^2} \\
\sigma_y^2 &= \frac{\sigma_z^2(\mu_u - c_x)(\mu_v - c_y) + \sigma_v^2(\mu_z^2 + \sigma_z^2)}{f_y^2} \\
\sigma_{xz} &= \sigma_{zx} = \sigma_z^2 \frac{\mu_u - c_x}{f_x} \\
\sigma_{yz} &= \sigma_{zy} = \sigma_z^2 \frac{\mu_v - c_y}{f_y} \\
\sigma_{xy} &= \sigma_{xy} = \sigma_z^2 \frac{(\mu_u - c_x)(\mu_v - c_y)}{f_x f_y}
\end{aligned}$$

The above are derived with μ_z and σ_z from the simple depth uncertainty model. We can then approximate Σ in terms of

the Gaussian mixture model by replacing μ_z and σ_z with $\widehat{\mu}_z$ and $\widehat{\sigma}_z$ respectively. We approximate $\mathbf{p}$ as a multivariate Gaussian distribution with mean $\mu = [\mu_x, \mu_y, \mu_z]^T$ and covariance Σ.

III. TRAJECTORY ESTIMATION

A. Overview

The trajectory estimation begins with extracting sparse features in each incoming RGB-D image I_t. The features are detected on the intensity channel of the RGB image. We have experimented with several choices of feature detectors, including SURF [1], ORB [9], and Shi-Tomasi [12] keypoints. While our implementation offers a configurable choice between them, we found that the Shi-Tomasi features offer the best trade-off between robustness and computational speed. Note that we only compute the 2D location of the keypoint in the image; we are not interested in the keypoint's local descriptor vector.

Next, we estimate the 3D normal distribution for each feature, according to the uncertainty equations defined in Sec. II. From this, we generate a set of 3D features $D_t = \{\mathbf{d}_i\}$. Each feature $\mathbf{d} = \{\boldsymbol{\mu}^{[D]}, \Sigma^{[D]}\}$ has a mean and covariance matrix. We label this set as the *Data*. The *Data* set is expressed in the camera reference frame.

We have a similar *Model* set, $M_t = \{\mathbf{m}_j\}$ with $\mathbf{m} = \{\boldsymbol{\mu}^{[M]}, \Sigma^{[M]}\}$ expressed in the fixed frame of reference. The *Data* set is aligned to the *Model* using ICP and rotated into the fixed frame. Next, we establish the final correspondences between the two sets. Re-observed features in the model are updated using a Kalman Filter, and new features are added to the previous model. The rest of this section describes the steps in more detail.

B. Registration

We define a distance function $dist$ which measures the distance between two features $\mathbf{f}_a$, $\mathbf{f}_b$, normally distributed with means $\boldsymbol{\mu}_a$ and $\boldsymbol{\mu}_b$ and covariance matrices Σ_a and Σ_b.

$$dist(\mathbf{f}_a, \mathbf{f}_b) = \sqrt{\Delta_{\mathbf{f_a f_b}} (\Sigma_a + \Sigma_b)^{-1} \Delta^T_{\mathbf{f_a f_b}}} \tag{9}$$

where

$$\Delta_{\mathbf{f_a f_b}} = \boldsymbol{\mu}_a - \boldsymbol{\mu}_b \tag{10}$$

The distance function is based on the Mahalanobis distance from a point to a distribution.

We use ICP to align the *Data* to the *Model* . As an initial guess for the ICP registration, we use the previous transform T_{t-1} between fixed and the camera coordinate frames.

The ICP algorithm has 2 steps of interest: generating correspondences between the two input sets, and calculating the transformation which minimizes the distance between the correspondences. In the classical ICP formulation, the correspondences are generated using nearest neighbors in Euclidean space; the transformation is also estimated by minimizing the sum of squared Euclidean distances.

We use a modified ICP algorithm, in which we establish approximate correspondences using the Mahalanobis distance. First, we build a kd-tree [6] of the *Model*, by using the means of the features. Next, for each feature $\mathbf{d}$ in the *Data* we find the k nearest Euclidean neighbors from the *Model*. Finally, we iterate through all k candidates, and find the one which has the smallest Mahalanobis distance. This allows us to leverage the efficiency of kd-trees, which cannot be directly used with non-linear functions such as the Mahalanobis distance. In our implementation, we use a small size for k (for example, 4).

The rest of the ICP algorithm remains the same. We note that while it is possible to optimize a Mahalanobis distance as the objective function for the best transform, we do not do so, and this is a possible area of improvement. An example of an algorithm which implements a similar optimization (albeit in the context of dense data) is Generalized ICP [11].

C. Data association and updating

We begin the data association step by rotating the *Data* set D into the fixed frame of reference, and refer to it as $D^{'}$. Let the current transformation between the fixed and camera coordinate frames is T, consisting of a rotation and translation:

$$T = \begin{bmatrix} \mathrm{R} & \mathbf{t} \\ 0 & 1 \end{bmatrix} \tag{11}$$

We can transform the mean vector and covariance matrix according to:

$$\boldsymbol{\mu}^{'} = \mathrm{R}\boldsymbol{\mu} + \mathbf{t} \tag{12a}$$

$$\Sigma^{'} = \mathrm{R}\Sigma\mathrm{R}^T \tag{12b}$$

Next, for each point $\mathbf{d}_i^{'}$ in the transformed *Data* set, we find the approximate nearest Mahalanobis neighbor $\mathbf{m}_j$ in the *Model*.

$$dist(\mathbf{d}^{'}, \mathbf{m}) = \sqrt{\Delta_{\mathbf{d}'\mathbf{m}} (\Sigma^{[M]} + \Sigma^{[D']})^{-1} \Delta^T_{\mathbf{d}'\mathbf{m}}}$$

We consider two points to be associated if the distance between them is lower than a threshold ϵ. Typical values for ϵ include 7.82 or 11.35. The two thresholds correspond to the 95% and 99% probability tests that the data point is sampled from the given model distribution.

Any features in $D^{'}$ which cannot be associated are inserted as new members in M. The model is bounded in size, so if the maximum allowed size is exceeded, we remove the oldest features in the model. This is achieved by using a ring-buffer implementation.

For each feature which is associated, we perform a Kalman Filter update. We treat the distributions $\mathbf{m}$ as the prior, and the distribution $\mathbf{d}$ as the observation.

The predicted distribution is the same as the prior state of the model at time $t-1$.

$$\tilde{\boldsymbol{\mu}}_t = \boldsymbol{\mu}^{[M]}_{t-1} \tag{13a}$$

$$\tilde{\Sigma}_t = \Sigma^{[M]}_{t-1} \tag{13b}$$

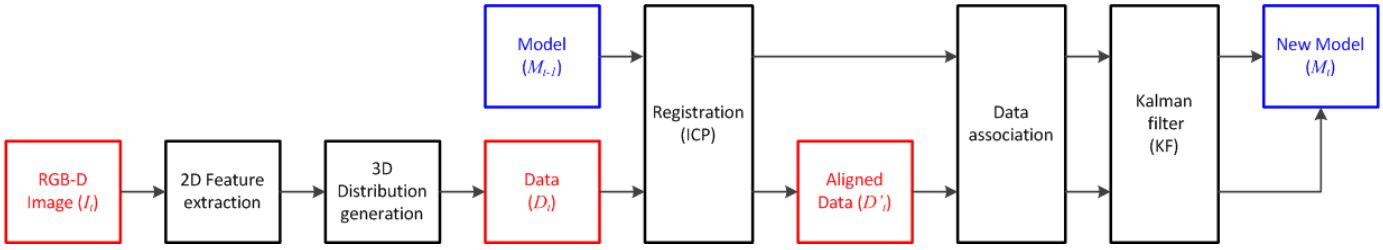

Fig. 3. Pipeline for the trajectory estimation. We align sparse feature data from the current RGB-D frame to a persistent model. The data is represented by 3D points with covariance matrices.

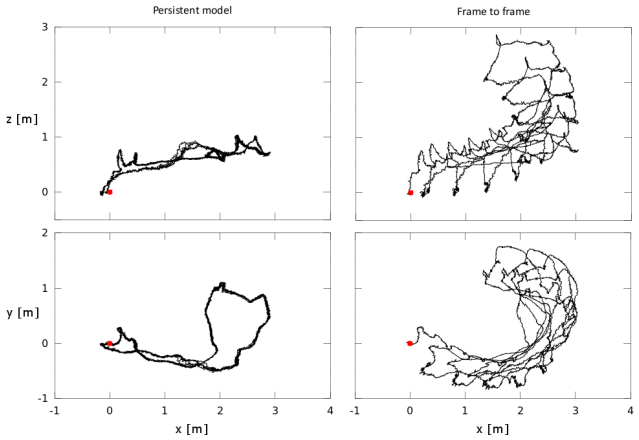

Fig. 4. Comparison of trajectory estimation with persistent model (left) vs frame-to-frame ICP. Top row: side views, xz-plane. Bottom row: top view, $xy-$ plane. The trajectory shown consists of 5 repeated loops, with approximately 2000 images processed in each loop.

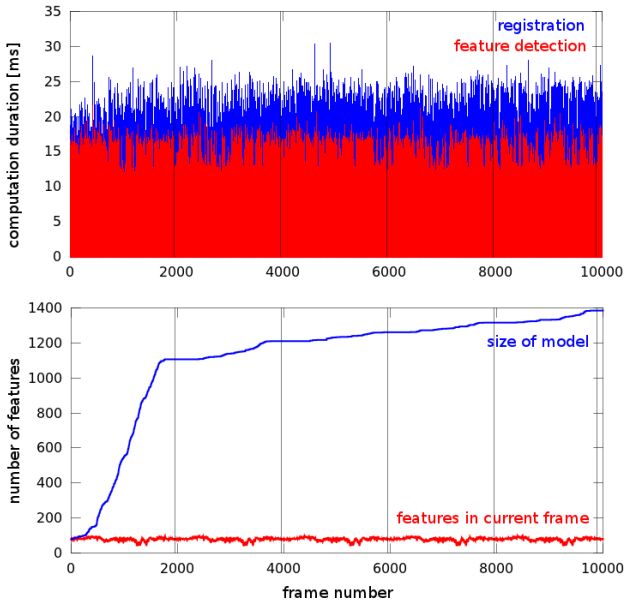

Fig. 5. Top: processing duration for each incoming image. Horizontal axis: image index number. Vertical axis: time required for feature detection (red) and alignment and model update (blue). Bottom: size of the *Data* and *Model* sets. Vertical lines mark the start of each repeated loop.

The following are the update (correction) equations for the new state at time t:

$$K_t = \tilde{\Sigma}_t \left(\tilde{\Sigma}_t + \Sigma_t^{[D]} \right)^{-1} \tag{14a}$$

$$\boldsymbol{\mu}_t^{[M]} = \tilde{\boldsymbol{\mu}}_t + K_t \left(\boldsymbol{\mu}_t^{[D]} - \tilde{\boldsymbol{\mu}}_t \right) \tag{14b}$$

$$\Sigma_t^{[M]} = (I - K_t)\, \tilde{\Sigma}_t \tag{14c}$$

IV. EXPERIMENTS

A. Visual odometry

We evaluate our trajectory estimation pipeline with RGB-D data recorded in an indoor environment. The camera is moved along a loop, and placed back at its starting point. Images are streamed at QVGA resolution. We replicate the data 5 times to simulate the exact same loop. Fig. 4 shows the trajectories generated with the persistent model (left), versus trajectories generated by frame-to-frame ICP (right). We show that our approach is able to correctly solve the loop, without any additional back-end optimization. Note that we do not claim that we can solve loops of arbitrary lengths; however, if that is needed, our approach can be combined with existing techniques such as pose-graph optimization or Sparse Bundle Adjustment.

Fig. 5 (top) shows the run time of the visual odometry. We divide the process in two parts: feature detection, and registration. The feature detection includes detecting the Shi-Tomasi corners and computing the distributions. The registration includes ICP run time and updating the model. Combined, they account for the entire processing time for each RGB-D image. We achieve an average processing time of 16.1ms, with a maximum of 30.5 ms and a standard deviation of 3.8ms.

Fig. 5 (bottom) shows the sizes of the *Data* and *Model* sets over time. We detect around 100 features in each frame. The model grows as new space is being explored. Once the loop is closed, number of new features inserted in the *Model* decreases significantly. We successfully close the loop with less than 1500 features in the *Model*.

B. Mapping

We implemented a mapping application which uses the output of the visual odometry and aggregates dense point

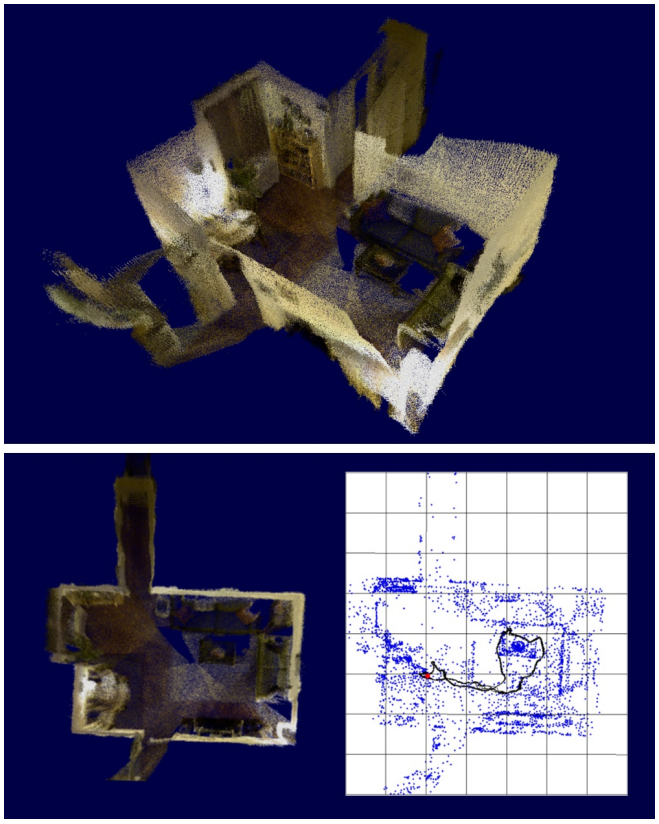

Fig. 6. Octree-based 3D map generated with our system. Bottom row shows the orthographic view of the dense 3D map (left), as well as the final *Model* set (right).

cloud data over time. The data is downsampled in real time using an octree to remove redundant points. It runs in a separate thread. The output map can be see in Fig. 6.

V. CONCLUSION

In this paper, we presented a visual odometry system for RGB-D cameras. The systems uses sparse features which are registered against a persistent model of bounded size. The model is updated through a probabilistic Kalman Filter framework. In order to achieve this, we developed a formulation for the 3D uncertainty in sparse features in RGB-D images, based on a Gaussian mixture model of readings in a local image window.

We have shown experimental results which demonstrate the predictive power of our uncertainty model, which is able to estimate uncertainties around object edges better than the previously published formulations in this field. We also present experimental results which demonstrate the speed and accuracy of our visual odometry.

Unlike many of the other state-of-the art systems, our algorithm does not use dense data, feature descriptor vectors, RANSAC alignment, or keyframe-based bundle adjustment. By avoiding these computations, we are able to achieve an average performance rate of 60Hz. Our visual odometry requires a single thread and does not need a GPU. The use of a persistent model, instead of frame-to-frame techniques, significantly decreases drift. We show that our method has sufficient accuracy to close loops in room environments natively, without any back-end post processing.

Several areas remain open to improvement. First, our probabilistic framework is used for the nearest neighbor association and model updates, but not in the actual ICP error minimization step. Second, we make the assumption that the depth image has zero-mean noise. However, we noticed that this assumption does not hold, and changing the camera affects the performance of the visual odometry. Thus, we need to further research methods of estimating and correcting the systematic error in the depth image. Finally, we would like to explore how our method can be applied in conjunction with classical loop-closing techniques, in order to be able to solve large-scale loops.

An implementation of our system, developed for use with the ROS framework, is available for download under a free, open-source license from our website (`http://robotics.ccny.cuny.edu/`).

REFERENCES

[1] Herbert Bay, Andreas Ess, Tinne Tuytelaars, and Luc Van Gool. Speeded-Up Robust Features (SURF). *Comput. Vis. Image Underst.*, 110(3):346–359, June 2008.

[2] Ivan Dryanovski, Carlos Jaramillo, and Jizhong Xiao. Incremental registration of RGB-D images. *Robotics and Automation (ICRA), 2012 IEEE International Conference on*, 2012.

[3] Felix Endres, Jürgen Hess, Daniel Cremers, and Nikolas Engelhard. An Evaluation of the RGB-D SLAM System. *Perception*, 3(c):1691–1696, 2012.

[4] Peter Henry, Michael Krainin, Evan Herbst, Xiaofeng Ren, and Dieter Fox. RGB-D Mapping: Using depth cameras for dense 3d modeling of indoor environments. In *In RGB-D: Advanced Reasoning with Depth Cameras Workshop in conjunction with RSS*, 2010.

[5] Kourosh Khoshelham and Sander Oude Elberink. Accuracy and Resolution of Kinect Depth Data for Indoor Mapping Applications. *Sensors*, 12(2):1437–1454, 2012.

[6] Marius Muja and D.G. Lowe. Fast approximate nearest neighbors with automatic algorithm configuration. In *International Conference on Computer Vision Theory and Application VISSAPP'09)*, volume 340, pages {331–340}. INSTICC Press, 2009.

[7] Richard A Newcombe, Andrew J Davison, Shahram Izadi, Pushmeet Kohli, Otmar Hilliges, Jamie Shotton, David Molyneaux, Steve Hodges, David Kim, and Andrew Fitzgibbon. KinectFusion: Real-time dense surface mapping and tracking. In *Mixed and Augmented Reality (ISMAR), 2011 10th IEEE International Symposium on*, pages 127–136, 2011.

[8] D Nister, O Naroditsky, and J Bergen. Visual odometry. In *Computer Vision and Pattern Recognition, 2004. CVPR 2004. Proceedings of the 2004 IEEE Computer Society Conference on*, volume 1, pages I–652 – I–659 Vol.1, 2004.

[9] E Rublee, V Rabaud, K Konolige, and G Bradski. ORB: An efficient alternative to SIFT or SURF. In *Computer Vision (ICCV), 2011 IEEE International Conference on*, pages 2564–2571, 2011.

[10] D Scaramuzza and F Fraundorfer. Visual Odometry [Tutorial]. *Robotics Automation Magazine, IEEE*, 18(4):80–92, 2011.

[11] A. Segal, D. Haehnel, and S. Thrun. Generalized-ICP. In *Robotics: Science and Systems*, Seattle, USA, 2009.

[12] Jianbo Shi and C Tomasi. Good features to track. In *Computer Vision and Pattern Recognition, 1994. Proceedings CVPR '94., 1994 IEEE Computer Society Conference on*, pages 593–600, June 1994.

[13] F Steinbrucker, J Sturm, and D Cremers. Real-time visual odometry from dense RGB-D images. In *Computer Vision Workshops (ICCV Workshops), 2011 IEEE International Conference on*, pages 719–722, 2011.

[14] Zhengyou Zhang. Iterative point matching for registration of free-form curves and surfaces, 1994.