LoopSmart: Smart Visual SLAM Through Surface Loop Closure

Guoxiang Zhang¹ and YangQuan Chen²

Abstract—We present a visual simultaneous localization and mapping (SLAM) framework of closing surface loops. It combines both sparse feature matching and dense surface alignment. Sparse feature matching is used for visual odometry and globally camera pose fine-tuning when dense loops are detected, while dense surface alignment is the way of closing large loops and solving surface mismatching problem. To achieve smart dense surface loop closure, a highly efficient CUDA-based global point cloud registration method and a map content dependent loop verification method are proposed. We run extensive experiments on different datasets, our method outperforms state-of-the-art ones in terms of both camera trajectory and surface reconstruction accuracy.

Keywords: Dense loop closure, SLAM, 3D reconstruction

I. INTRODUCTION

Our goal is to track a RGB-Depth (RGB-D) camera accurately in real time and optimize its trajectory when new information is available. Along with recovering camera trajectory, we also want to get globally consistent 3D models. This falls into the research area of visual simultaneous localization and mapping (SLAM). With both accurately estimated camera pose and 3D dense map, an SLAM system is very useful for a broad spectrum of applications, such as robot navigation, 3D dataset generation [1], [2], digital heritage [3] and augmented reality.

There are still many challenges in making a well working SLAM system which emphasizes both camera trajectory and dense surface estimate accuracy, even though it has been actively researched for many years [4]. Major difficulties are from three sources. First, the camera tracking error accumulation problem can be considered unavoidable due to the incremental nature of any SLAM system. Second, it is hard to find all important loops. Intuitively, the more loops in data, the more information can be used to recover precise camera trajectory and the 3D model. But, in practice, accuracies of separating dense models.

To get a better camera tracking and dense mapping accuracy, researchers tried different ways to attack this problem. For a better tracking accuracy, in [5], curvature information was added into the frame-to-model iterative closest point (ICP). To better close those loops, most SLAM systems [6], [7], [8] use bag of words (BoW) [9], but it is well known that it is not very reliable under certain lighting condition or viewing angle changes. Since the BoW only matches sparse features from images but cannot fully utilize all camera observation data and spatial priors. On the other hand, SLAM systems tend to add the loops very conservatively so as to reduce severe influence of the false loops, thus many important loops may not be connected. Even after loops are successfully detected, there is still another problem in the dense SLAM system: how to correct reconstructed surface optimally. Since most dense visual SLAM systems [10], [6], [11] use a frame-to-model fusion process, which makes it difficult to quantify, isolate and remove the influence of past camera data, and it is also computational expensive for a full camera data sequence re-fusion. [11] suggested to form a deformation graph across the reconstructed dense model to deform its surface to connect the loop. When loops become large, the model may not be deformed optimally, since past camera observation are not reused to manipulate the 3D model. They assumed that the scenes are elastic, but in reality, they are mostly rigid.

Motivated by the fact that human can notice mismatches in dense 3D models very easily by just looking at the spatial displacement of surfaces. We propose to resolve mismatches directly by closing dense loops to get a consistent 3D model and a precise camera trajectory estimate in the visual SLAM system. After dense loops detected, instead of optimizing dense surface directly to propagate correction introduced by dense loop, in this paper, dense loops correction is done through sparse feature bundle adjustment, so that all the past camera poses can be corrected based on their observations. By running extensive experiments on different datasets, we observed that combining sparse feature with dense loop closure can produce state-of-the-art performance not achieved before. Not only 3D model gets improved, but also camera trajectory estimate becomes more accurate. This is because our framework can detect loops in dense surface domain and optimize loops in sparse feature domain. Note that our framework can detect dense loops, yet other other means of detecting loops can still be utilized.

In the following, we summarize the key contributions of our method:

1) We are the first to correct dense loops with sparse feature based bundle adjustment in a visual SLAM

This work was supported by CITRIS seed grant titled SmartCaveDrone, NVIDIA GPU grant and UC Merced Bobcat award.

¹Mechatronics, Embedded Systems and Automation Lab, School of Engineering, University of California, Merced, Merced, CA, USA, gzhang8@ucmerced.edu
²Mechatronics, Embedded Systems and Automation Lab, School of Engineering, University of California, Merced, Merced, CA, USA, ychen53@ucmerced.edu
system. We demonstrate this dense-sparse combination can give much improved camera tracking and dense modeling results.

2) We propose and implement a CUDA based global point cloud registration algorithm that is fast and accurate so that it can be used in a real-time SLAM system.

3) We can correct dense surface in real time based on the motion of camera trajectory estimate.

II. RELATED WORK

Visual SLAM has been studied actively for many years. Researchers from different fields, such as robotics, computer vision and computer graphics, tried to solve this problem with their own emphases and preferences, which lead to diverse visual SLAM systems.

Sparse feature based SLAM systems are well developed, because sparse features can be used to sample data from sensor reading (e.g. images) to sparse data representation as image keypoints and feature vectors, which means less computation, since data from different frames is matched solely based on feature vectors of their keypoints. Extended Kalman filter or particle filter based filtering approaches [12], [13] can take keypoints as visual landmarks and solve visual SLAM as a data filtering process. A drawback of this approach is that the filter cannot be re-optimized again based on all previous data. Then, maximum a posteriori (MAP) based approaches are used to optimize all observed camera data in a batch setting [8], which utilizes bundle adjustment (BA) from Structure from Motion [14], [15], to get a better accuracy. They usually have local BA in camera tracking to have map size independent time complexity [8]. In order to run BA for loop closure, the loops need to be detected. In the sparse image feature setting, BoW based loop closure is widely used. But it can give high portion of false loops, which can severely degrade performance of a SLAM system, so a very strict loop filtering is often used [8], where many loops are blindly rejected. This causes a big problem when there are many loopy motion in camera movement.

Another line of visual SLAM research tried to focus on dense surface reconstruction. With the parallel processing power of GPU, Newcombe et al. proposed KinectFusion [10] which performs real-time dense 3D camera tracking and model fusion. It has a volumetric scene representation, which can be rendered to a depth map at a given camera pose. Tacking is done through a frame-to-model projective ICP, which is parallelized on GPU for real-time performance. Finally, new camera data is fused into the volumetric model using running average. KinectFusion can be considered of fusing very local loops together using the model it maintains as a proxy, but it does not close large loops. To close large loops, how to detect loops and find relative poses between loop areas need to be solved. In BoW, image keypoints and feature are used for both loop detection and relative pose generation, but in dense 3D systems, there is no such comparably reliable point cloud feature. Whelan et al. use BoW in a dense SLAM system called Kintinuous [6], which is an extended KinectFusion system. Later, to better solve loop detection and optimization problem, ElasticFusion [11] proposed to use ICP to find relative poses of potential loops, which are proposed by two sources of information: spatial prior and appearance-based place recognition. Our work shares similarity with [11] in using the spatial prior, but we propose a totally different approach and underlying algorithms. Instead of using projective ICP which highly depend on initialization. We propose a GPU based global point cloud registration method to close loops with another prior information from sparse feature alignment. After dense loop detected, in Kintinuous, a pose graph of keyframes was utilized, while the authors mentioned that mesh deformation was required to get smooth 3D models, which indicates loop correction is not done optimally. In ElasticFusion, the pose graph is totally replaced by a deformation graph distributed inside the dense model. This deformation graph does not have a backing physical meaning, because most of scenes scanned are not elastic. In our framework, we utilize bundle adjustment to have an MAP correction of all past keyframes, which is theoretically optimal. BundleFusion [15] used bundle adjustment to optimize loops, but they do not close dense loops, instead, they close sparse feature loops and only use dense surface for feature correspondence search and tracking. Also, they use the idea of re-integrating camera data, which consumes much more computation than our approach.

III. FRAMEWORK OVERVIEW

We divide our framework to four major parts: 1) tracking, 2) dense model fusion, 3) dense loop detection and verification, 4) camera poses optimization and map correction. They are tightly coupled, as in Fig. 1 so that more information can be exploited out of sensor data.

**Tracking.** We employ the tracking part from ORB-SLAM2 [8], which is a very well implemented sparse feature based SLAM system. Inside this tracking module, Oriented FAST and Rotated BRIEF (ORB) feature is extracted for keypoint matching. Then frames are tracked against keyframes with motion estimate and then refined with the local sparse
map. Keyframes are generated when tracking is weak or local bundle adjustment thread is free. Local BA is used to correct re-projection error of feature correspondences among co-visible keyframes in a background thread.

**Dense model fusion.** We fuse dense models on a GPU using surfels as a map representation similar to [16], [11]. Each surfel has a position $p$, normal vector $n$, radius $r$, confidence $c$, initialization time stamp $t_0$, last updated time stamp $t_u$. It has a sparse data structure because it only stores data where surface exists, thus it is suitable for larger scene. More importantly, when the camera trajectory gets optimized, it is easier to manipulate a surfels model than a model with volumetric representation. We fuse every $k$ frames ($k = 50$ for all experiments, as in [17]) into a fragment of dense model, called scene fragment. These scene fragments are generated for two reasons. One is to integrate out raw RGB-D data noise. Another one is to reduce the number of 3D pieces so that loop detection computation are accelerated. After each scene fragment is generated, it is linked to the keyframe whose time stamp is closest to the ( $\frac{1}{k}$)-th frame, for the purpose that scene fragments can be transformed accordingly when camera trajectory estimate is updated.

**Dense loop detection and verification.** In order to close dense loops effectively and efficiently, we utilize strong prior and CUDA accelerated computing. First, only the scene fragment pair with overlapping in camera observation can be a loop candidate. Second, we use a co-visibility graph to pre-filter loop proposals that are co-visible. Then filtered loop proposals are aligned using a CUDA based global point cloud registration method we proposed. If two point cloud can be aligned together, the loop proposal becomes a loop candidate with a relative transformation matrix that can align them together. Finally, the loop candidates need to go through a loop verification process, so that false loops would not diverge the subsequent optimization process.

**Loop optimization and correction.** After dense loops detected and verified, they are used to connect pose graph and also trigger more image loop detection, which again uses spatial prior and ORB feature matching. Then, the pose graph is optimized to give a coarse pose correction and then a full BA is performed in order to get MAP optimally fine-turned camera trajectory estimate. Finally, both spare map and dense model are corrected based on the update of camera trajectory estimate.

### IV. FAST GLOBAL POINT CLOUD REGISTRATION ON GPU

To formulate dense loop closure formally, we denote a dense scene fragment as a set of points $P$ with their normal. Then the dense fragment based loop closure problem can be solved by point cloud registration methods. In a SLAM system, this registration needs to be done very quickly to meet the real-time criteria. In ElasticFusion, Whelan et al. resort to projective ICP, which can be performed very quickly on GPU. But, when mismatches are big, the initialization-dependent nature of ICP makes it difficult to converge to the right solution. So we turn to global point cloud registration in our framework, which does not depend on initial alignment at all. From evaluation in [18], global registration methods are either too slow or have unreliable performance. We find the one [17] with best recall can be parallelized and accelerated using GPU computing so that it can be both the most accurate and fastest.

Our algorithm is based on RANSAC and inspired by [17], which is modified from Point Cloud Library [19]. The major differences are that we accelerated the most time consuming parts using GPU programing with an efficient nearest neighbor search method, and that normal checking is moved from pre-rejection part into hypotheses testing. Traditionally, RANSAC is formulated as an iterative process.

#### Algorithm 1 Global Registration on GPU

| Input: A pair of point cloud $P$ and $Q$ |
| Output: $T_{QP}$ if $P$ and $Q$ can be aligned together; |
| Downsample $P$ and $Q$; |
| Compute FPFH features $F(Q)$ and $F(Q)$; |
| $T_{QP} \leftarrow I$; |
| $feature_{NN} \_cache \leftarrow \emptyset$ |
| Quadruple point set samples $S_P \leftarrow \emptyset$, $S_Q \leftarrow \emptyset$ |
| // Parallel FPFH feature pre-matching |
| for $i \leftarrow 1$ to $P_{\_count}$ do on GPU cores in parallel |
| $feature_{NN} \_cache[i] \leftarrow NN$ of $F(p_i)$ in $F(Q)$; |
| end for |
| // Randomly sample hypotheses on GPU |
| for $i \leftarrow 1$ to $sample\_number$ do on GPU cores in parallel |
| $S_P[i] \leftarrow$ randomly picked ($p_0, p_1, p_2, p_3$) from $P$; |
| for all ($p_0, p_1, p_2, p_3$) do |
| $q_j \leftarrow$ $NN$ of $p_j$ in $Q$ using $feature\_NN\_cache[j]$ |
| end for |
| $S_Q[i] \leftarrow (q_0, q_1, q_2, q_3)$ |
| // Hypotheses pre-rejection |
| for all ($p_0, p_1, p_2, p_3$), ($q_0, q_1, q_2, q_3$) do |
| if $||p_k - q_{k+1}|| < \tau ||q_k - q_{k+1}||$ or vice versa then |
| set current hypothesis pre-rejected in $S_P$ and $S_Q$; |
| end if |
| end for |
| // Consensus for the remaining quadruple pairs |
| for $i \leftarrow 1$ to $remain\_number$ do on GPU cores in parallel |
| Estimate transformation $T_{QP}$ from $S_P[i]$ to $S_Q[i]$; |
| Find all inliers under $T_{QP}$ hypothesis; |
| $test\_log \leftarrow$ inlier ratio, fitness score |
| end for |
| // Get final result from $test\_log$ |
| Find the Hypothesis that has max inlier ratio and fitness score passing minimum threshold in $test\_log$. |
| if Hypothesis found then |
| return its $T_{QP}$; |
| end if |
with proved convergence [20]. But, if we relax the theoretical convergence requirement, different iterations and different hypotheses can be considered to be totally independent to each other. This means different hypotheses can be mapped to different processing cores to be tested in parallel.

As shown in Algorithm 1 point clouds are downsampled to the resolution of the typical precision of RGB-D sensors to reduce unnecessary computation. Fast Point Feature Histograms (FPFH) features are extracted for each point in P and Q for point correspondence pairs generation. To make nearest neighbor search more efficient, we pre-cached all the nearest neighbors of P in Q using FPFH feature distance. Then, for each hypothesis, 4 points are randomly sampled from P, and their correspondences are found through the pre-cached nearest neighbors. After that, a pre-rejection step, which rejects hypotheses whose point pairs cannot make a similar polygon, is performed. \( \tau \) is a similarity threshold and set to 0.9 in all our experiments. Then, hypotheses testing, the most time consuming step, tests both inlier ratio and fitness score on GPU. When implement it, we test each hypothesis on a thread block with efficient parallel reductions. For the NN search during hypotheses testing, we utilized a 3D grid to replace the k-d tree to fit special need of a GPU, since a GPU will slow down when different threads go to different code branches during k-d tree search. We propose to use a 3D grid for NN search, given that the point clouds to be searched only span in a limited area. This guarantees that we can use a grid with limited size for NN searching without jeopardizing searching accuracy. When a point is stored into the search grid, the indices of its cell is calculated by eq. (1).

\[
\begin{align*}
index_x &= \frac{(x_p - x_{center})}{cell length}, \\
index_y &= \frac{(y_p - y_{center})}{cell length}, \\
index_z &= \frac{(z_p - z_{center})}{cell length},
\end{align*}
\]

where the \( x_{center}, y_{center}, z_{center} \) are the coordinate of the center of target point cloud. \( cell length \) is the cell edge length of the NN 3D grid. It is subtracted so that the translation of point cloud does not affect searching. When a point wants to query its nearest neighbor, the searching is accomplished through table looking up, which has a time complexity of \( O(1) \), given cell edge length the same as point cloud downsampling resolution. It is faster than the k-d tree which has a \( O(\log n) \) time complexity. We observed speedup by only replacing k-d tree with 3D search grid in CPU only code. More importantly, there is no branching during searching, so it fits much better on a GPU than the k-d tree.

We run experiments to compare speed, recall and precision performance against two baseline methods: CZK [17] and FastGlobalRegistration (FGR) [18] on redwood pairwise registration evaluation dataset by Choi et al. We report results in Table I. Recall and precision for CZK are from reports of paper authors, but speed is measured on our machine. Not all the complete results of FGR is publicly available from its authors, so we measure them on our own. For all methods, we use the same hyper-parameter values as in published code of [17]: 0.05 m as point cloud downsampling leaf size, 0.1 m as normal estimate radius, 0.25 m as FPFH feature estimate radius, 4000000 as hypotheses count and 0.075 m as maximum point correspondence distance. We use an Intel i7-6850K clocked at 3.6 GHz and an NVIDIA Titan X Pascal for our evaluation.

Our global point cloud registration can finish in around 20 milliseconds, which is around 366 times faster than CZK and 5 times faster than FGR as in Table I. With this speed, it can run at 50 Hz, which means we can process more loop candidates. Our recall performance is not compromised for speed, instead, it is even better than all these two methods. Even though it is much faster than other methods, it is necessary to figure out a smart way of using our point cloud registration method for a high performance SLAM system. So we propose ways to pre-filter unnecessary loop detection using spatial priors. From Table I the low precision is an issue needs to be solved, so an effective loop verification method is also adapted. We describe these in the following section.

V. DENSE LOOP PROPOSAL AND POST-FILTERING

To avoid unnecessary loop detection computation, we use strong spatial priors and connections between dense model and sparse map to generate loop proposals and then only run global point cloud registration on these loop proposals. The first criteria of dense loop proposal is spatial overlapping of scene fragments pair in their camera observations. This is based on the prior that mismatch is more likely to happen when two surfaces have overlapping. One may argue that this will limit the scope of loops can be closed, but given indoor environment is usually small and tracking model from ORB-SLAM2 is considerably accurate, we do not observe any far away drift without any surface overlapping. If that could happen, we can easily solve it by adding image based place recognition to generate more spatial location independent loop proposals, which fits perfectly with the rest of the processing pipeline.

There is more information can be exploited to filter these loop proposals, because it is not always necessary to close dense loops when sparse feature tracking itself has tracked
new area into the old map. Or, in the case that, after closing one dense loop, sparse feature tracking successfully find sparse map connection between new map and old map, then there is no need to continue detecting dense loops. This makes our dense loop detection smart, so it only run detection when necessary.

All the loop proposals will be fed into GPU based point cloud registration to recover relative pose of aligning two point clouds, if they can be aligned. As mentioned before, the low precision of a global registration method is a serious problem, so we add a loop post-filtering process. Our idea is to utilize information of pose graph. Because when doing experiments, we find false loop usually bring in dramatic changes of pose graph structure, while true positive loops make just small correction. Again, this assumption applies when the tracking model is reliable enough, which holds in our case.

To formulate this prior information mathematically, we turned to line processes based robust optimization proposed in [17]. Our adapted objective function is eq. (2):

$$E(\tau, I) = \lambda \sum_i f(T_i, T_{i+1}) + l_{ij}(f(T_i, T_j, T_{ij}) + \mu \Psi(l_{ij}),$$

where

$$f(T_i, T_j, T_{ij}) = \sum_{(p, q) \in K_{ij}} \| T_i p - T_j q \|^2,$$

$$\Psi(l) = (\sqrt{l} - 1)^2,$$

same as in [17], $T_i$ is the i-th camera pose in 4-by-4 matrix format, $T_i$ is a matrix that transform points from the coordinate system of $P_i$ to the coordinate system of $P_j$, $K_{ij}$ is the set of nearest neighbor correspondence pairs between $T_i P_i$ and $T_j P_j$ that are within distance $e = 0.05$ m, which is typical noise level of RGB-D sensor. $\lambda$ is weighting factor of odometry term and is set to 1000 in all our experiments. $\mu = \tau k$, and $\tau = 0.2$ is a distance threshold. Weight $l$ which controls the influence of the loop candidate tested, is optimized during iterations. To accelerate this computation, eq. (5) is formulated to approximate eq. (3) in [17]:

$$f(T_i, T_j, T_{ij}) = \xi^T \sum_{(p, q) \in K_{ij}} G^T G \xi,$$

where $\xi$ is a 6D rotation and translation vector $(\alpha, \beta, \gamma, x, y, z)$ to represent matrix $T_i, T_j, T_{ij}, G_{pq} = [-[p], [q]]$. With this approximation, this objective function is used solving g2o. A loop proposal is rejected when $l < 0.25$. We periodically (every 50 fragments in all experiments) update $G^T G$ so that our graph is not outdated.

VI. LOOP OPTIMIZATION AND CORRECTION

After a dense loop passing verification, map optimization and correction are followed to reduce mismatches and error. We employ BA to get a MAP based optimization to correct the camera trajectory estimate and the dense model. During optimization, model area with earlier time stamp is fixed as a target. The rest model is moved to match it. First, the key frame associated with the source scene fragment get transformed into coordinate system of target scene fragment using the relative pose $T$. Its co-visible keyframes are also transferred using $T$. Then more image based loop detection is triggered to detect more loops. After detecting new image feature loops, similar to [8], a pose graph optimization is performed to give a rough camera pose trajectory and then bundle adjustment is performed by optimizing eq. (6):

$$\min \sum_c \sum_{p \in V_c} \rho \left( \| \tilde{s}_p - K[R_c[T_i]X_p] \|_2 \right),$$

where $\rho$ is the robust Huber cost function, $\tilde{s}_p$ is the observed 2D pixel location in the camera coordinate system, $K$ is the camera intrinsics matrix, $R_c$ and $T_i$ are the camera orientation and position of c-th keyframe pose $T_c$, $X_p$ is the location a point visible from the c-th camera, $\Sigma$ is the covariance matrix associated to the scale of the keypoint.

Since we have a front end of displaying real-time dense surface, it is important to have dense model corrected in real time as well. Previously, [15] uses an idea of de-fusion and reintegrate on a volumetric representation, but, as mentioned before, de-fusion process cannot really reverse the fusion process, due to its filtering nature. In ElasticFusion, the surface is deformed based on a deformation graph. Here, we propose a novel way of correct dense surface based on the correction motion of camera pose estimate. The idea is to move the dense surface with observing keyframe cameras. When pose corrections are applied to a keyframe, we move all the surfels it can observe together with this correction. To apply this idea, two problems need to be dealt with:
1) Most surfels are observed from adjacent camera poses;
2) Surfels from overlapping area can be observed from two distinctly different camera sub-trajectory, i.e. the trajectory in loop area. We solve the first problem by averaging the influence from adjacent keyframe camera poses. Our solution to the second one is to take the influence of keyframe set $F$ only when $t_0 < t_{keyframe} < t_a$, so that only keyframes in the time window between initial and updating are considered. The updated position and normal of a surfel is formulated as eq. (7)

$$p_{updated} = \frac{1}{k} \sum_{f \in F} T_{f, new} T_{f, old}^{-1} p_{old},$$

$$n_{updated} = \text{normalize} \left( \frac{1}{k} \sum_{f \in F} R_{f, new} R_{f, old}^{-1} n_{old} \right).$$

We also remove points who are not observed from any camera pose. This dense model correction can be done very efficiently on GPU, since each surfel only take influence from very limited numbers of keyframes and there is no inter-surfel dependence.

VII. RESULTS

We have evaluated our LoopSmart system on different datasets and compare it with other online and offline meth-
TABLE II: Surface reconstruction error (in meters) on augmented ICL-NUIM sequences

|          | Living room 1 | Living room 2 | Office 1 | Office 2 | Average |
|----------|---------------|---------------|----------|----------|---------|
|          | mean | median | mean | median | mean | median | mean | median | mean | median |
| ElasticFusion | 0.084 | 0.050 | 0.111 | 0.062 | 0.081 | 0.036 | 0.043 | 0.019 | 0.080 | 0.042 |
| BundleFusion | 0.061 | 0.020 | 0.065 | 0.022 | 0.016 | 0.053 | 0.030 | 0.019 | 0.043 | 0.029 |
| CZK | 0.035 | 0.019 | 0.028 | 0.019 | 0.019 | 0.015 | 0.022 | 0.014 | 0.026 | 0.017 |
| ORB-SLAM odometry | 0.031 | 0.015 | 0.022 | 0.018 | 0.019 | 0.012 | 0.014 | 0.010 | 0.022 | 0.014 |
| ORB-SLAM full | 0.017 | 0.010 | 0.010 | 0.008 | 0.015 | 0.010 | 0.013 | 0.010 | 0.014 | 0.010 |
| Ours | 0.007 | 0.005 | 0.007 | 0.006 | 0.013 | 0.009 | 0.010 | 0.007 | 0.009 | 0.006 |

Fig. 2: Distance error map of reconstructed models from different methods against ground-truth on Living room 1 data sequence.

TABLE III: RMSE error (in meters) of estimated camera trajectories

|          | Living room 1 | Living room 2 | Office 1 | Office 2 |
|----------|---------------|---------------|----------|----------|
| ElasticFusion | 0.50 | 0.36 | 0.14 | 0.10 |
| BundleFusion | 1.26 | 0.64 | 2.01 | 0.12 |
| CZK | 0.10 | 0.13 | 0.06 | 0.07 |
| ORB-SLAM2 odometry | 0.14 | 0.05 | 0.05 | 0.03 |
| Full ORB-SLAM2 | 0.10 | 0.03 | 0.04 | 0.03 |
| Ours | 0.04 | 0.02 | 0.03 | 0.02 |

From Table II and Table III our system can give best results on all data sequence in terms of both trajectory and surface estimation accuracy. To give a more informative
Fig. 3: Error map and reconstructed model comparison on Living room 2 data sequence.

comparison, we report error map of reconstructed model in Fig. 2 on Living room 1 data sequence. We can see our result have lowest error across the whole model. Other than Living room 1 sequence, ORB-SLAM2 performs also very well. To better understands the difference inside these cases, we report error map and reconstructed model in Fig. 3 on Living room 2 data sequence. In Fig. 3 even though two reconstructed models looks very similar, but the error maps shows our system produce more accurate result.

B. Real-world scenes

We run experiments on public real-world datasets for a qualitative analysis and a robustness test. We run our system on Copyroom and Lounge dataset from Zhou et al. and DysonLab dataset from Whelan et al. We report our qualitative results in Fig. 4. Since there are no ground-truth models available, we only report screen-shot of reconstructed model from our system. Due to space limitation, we do not include results of other systems. Interested reader can refer to authors release. Visually, our system produces results at least matching the state-of-the-art methods.

We also run experiments on a dataset collected by a team of archaeologists. This dataset is collected in Maya caves using a RGB-D sensor Kinect V1 at Belize. LED lights are used to light up the environment as in Fig. 5 and caves are scanned with loopy motion. We compared results from ORB-SLAM2, ElasticFusion and our system. Note that we set ICP weight to 80 for ElasticFusion to get better tracking. The ORB-SLAM2 suffers from closing loops due to lighting changes and appearance similarity across frames. We show typical results in Fig. 6. Our system can produce more consistent 3D models, thus get better performance than other systems. When compared with ORB-SLAM2, it shows the importance of closing dense loops. Compared with ElasticFusion, it shows the advantages of having global registration and bundle adjustment as the way to optimize loops.

VIII. CONCLUSIONS

We demonstrate that the LoopSmart framework, proposed of combining dense loops with sparse feature optimization, delivers state-of-art performance. We also want to emphasize that our dense visualization part is optional, so it is possible to further improve performance to make it available on an embedded system, such as an NVIDIA TX 2.

IX. FUTURE WORK

We target this paper at the framework level innovation to introduce dense loop into sparse feature based tracking and optimization SLAM system, so we put together a working
system which enable us to test this framework. It is possible to improve submodules to further improve final results. For example, the tracking module can combine both sparse feature matching and dense ICP based approach to improve robustness at feature-less area. The dense loop verification module can be improved by add hard constraints of angular and positional movement during optimization. We put them in our future work to further improve our system.

ACKNOWLEDGMENT
We thank Mighty Chen and Kevin Ong for their help in converting format of RGB-D data.

REFERENCES
[1] J. Xiao, A. Owens, and A. Torralba, “SUN3D: A database of big spaces reconstructed using SfM and object labels,” in Proc. of the 2013 IEEE International Conference on Computer Vision, Dec 2013, pp. 1625–1632.
[2] A. Dai, A. X. Chang, M. Savva, M. Halber, T. Funkhouser, and M. Nießner, “ScanNet: Richly-annotated 3D Reconstructions of Indoor Scenes,” in Proc. Computer Vision and Pattern Recognition (CVPR), IEEE, 2017.
[3] G. Zhang, B. Shang, Y. Chen, and H. Moyses, “SmartCaveDrone: 3D cave mapping using UAVs as robotic co-archaeologists,” in Proc. of the 2017 International Conference on Unmanned Aircraft Systems (ICUAS), June 2017, pp. 1052–1057.
[4] C. Cadena, L. Carlone, H. Carrillo, Y. Latif, D. Scaramuzza, J. Neira, I. Reid, and J. Leonard, “Past, present, and future of simultaneous localization and mapping: Towards the robust-perception age,” IEEE Transactions on Robotics, vol. 32, no. 6, p. 13091332, 2016.
[5] D. Lefloch, M. Kluge, H. Sarbolandi, T. Weyrich, and A. Kolb, “Comprehensive use of curvature for robust and accurate online surface reconstruction,” IEEE Transactions on Pattern Analysis and Machine Intelligence, vol. PP, no. 99, pp. 1–1, 2017.
[6] T. Whelan, M. Kaess, M. Fallon, H. Johannsson, J. Leonard, and J. McDonald, “Kintinuous: Spatially extended KinectFusion,” in Proc. of RSS Workshop on RGB-D: Advanced Reasoning with Depth Cameras, Sydney, Australia, Jul 2012.
[7] F. Endres, J. Hess, J. Sturm, D. Cremers, and W. Burgard, “3-D mapping with an RGB-D camera,” IEEE Transactions on Robotics, vol. 30, no. 1, pp. 177–187, 2014.
[8] R. Mur-Artal and J. D. Tardós, “ORB-SLAM2: An Open-Source SLAM System for Monocular, Stereo, and RGB-D Cameras,” IEEE Transactions on Robotics, vol. PP, no. 99, pp. 1–8, 2017.
[9] D. Gálvez-López and J. D. Tardós, “Bags of binary words for fast place recognition in image sequences,” IEEE Transactions on Robotics, vol. 28, no. 5, pp. 1188–1197, October 2012.
[10] R. A. Newcombe, S. Izadi, O. Hilliges, D. Molyneaux, D. Kim, A. J. Davison, P. Kohi, J. Shutton, S. Hodges, and A. Fitzgibbon, “KinectFusion: Real-time dense surface mapping and tracking,” in Proc. of the 2011 10th IEEE International Symposium on Mixed and Augmented Reality, Oct 2011, pp. 127–136.
[11] T. Whelan, S. Leutenegger, R. S. Moreno, B. Glocker, and A. Davison, “ElasticFusion: Dense SLAM Without A Pose Graph,” in Proc. of Robotics: Science and Systems, Rome, Italy, July 2015.
[12] A. J. Davison and N. Kita, “3D simultaneous localisation and map-building using active vision for a robot moving on undulating terrain,” in Proceedings of the 2001 IEEE Computer Society Conference on Computer Vision and Pattern Recognition, CVPR 2001, vol. 1, 2001, pp. I–384–I–391 vol.1.
[13] M. Montemerlo and S. Thrun, “Simultaneous localization and mapping with unknown data association using FastSLAM,” in Proc. of the 2003 IEEE International Conference on Robotics and Automation (Cat. No.03CH37422), vol. 2, Sept 2003, pp. 1985–1991 vol.2.
[14] R. O. Castle, D. J. Gawley, G. Klein, and D. W. Murray, “Towards simultaneous recognition, localization and mapping for hand-held and wearable cameras,” in Proceedings 2007 IEEE International Conference on Robotics and Automation, April 2007, pp. 4102–4107.
[15] A. Dai, M. Niessner, M. Zollhofer, S. Izadi, and C. Theobalt, “BundleFusion: Real-time globally consistent 3D reconstruction using on-the-fly surface reintegration,” ACM Trans. Graph., vol. 36, no. 3, pp. 24:1–24:18, May 2017.
[16] M. Keller, D. Lefloch, M. Lambers, S. Izadi, T. Weyrich, and A. Kolb, “Real-time 3D reconstruction in dynamic scenes using point-based fusion,” in Proc. of the 2013 International Conference on 3D Vision: 3DV 2013. IEEE, 2013, pp. 1–8.
[17] S. Choi, Q. Y. Zhou, and V. Koltun, “Robust reconstruction of indoor scenes,” in Proc. of the 2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), June 2015, pp. 5556–5565.
[18] Q.-Y. Zhou, J. Park, and V. Koltun, “Fast Global Registration,” in Proc. of the European Conference on Computer Vision. Springer, 2016, pp. 766–782.
[19] R. B. Rusu, N. Blodow, and M. Beetz, “Fast point feature histograms (FPFH) for 3D registration,” in Proc. of the 2009 IEEE International Conference on Robotics and Automation, May 2009, pp. 3212–3217.
[20] M. A. Fischer and R. C. Bolles, “Random sample consensus: A paradigm for model fitting with applications to image analysis and automated cartography,” Commun. ACM, vol. 24, no. 6, pp. 381–395, June 1981.