Kinect Fusion

Reimplementation

Kerem Yildirir, Marc Benedi San Millan, Yigit Aras Tunali, and Poyraz Kivanc Karacam

"Team 18"

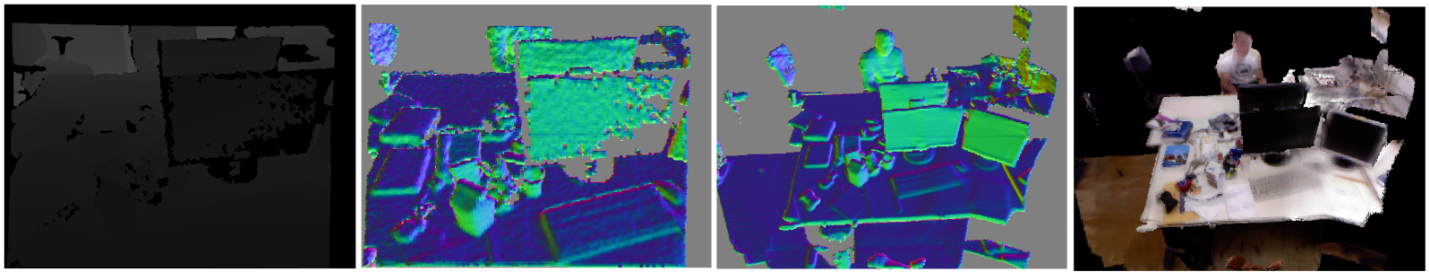

Figure 1: Example output of our method generated using the *freiburg1_xyz* dataset [5]. From left to right: First depth input from the sensor, normal vector reconstruction after processing the first frame, normal vector reconstruction of the model after processing 796 frames, and RGB reconstruction of the model after processing 796 frames.

---

1 Abstract

We re-implement the solution proposed by *Izadi, S. et. al.* [2]. It is a method for real-time large-scale indoor dense reconstruction, and is highly applicable in many areas such as Robotics, Augmented Reality and Human-Computer Interaction. We use the proposed 4 step pipeline with a few changes. Even though we were not able to capture 30FPS real-time performance, we successfully reconstruct various scenes using the proposed methods.

2 Introduction

In the field of Computer Vision, tracking of camera and simultaneously mapping of the physical scene is a problem that have been studied thoroughly. Approaches such as structure from motion (SFM) have been used to create a sparse representation and multi view stereo (MVS), but their focus was not real-time performance. In Kinect Fusion [2], it is showed that a robust real-time performance was possible, using a low cost depth sensor Microsoft Kinect and leveraging the performance of GPUs. This approach paves way for exciting new developments in the field of Augmented Reality with the example of Geometry aware AR in [2] and Robotics with it's real time performance and dense reconstruction.

3 Related work

Before this paper, there were other real-time approaches for the problem of real-time tracking and mapping PTAM [3] [4]. PTAM introduces the notion of parallelizing tracking and mapping processes. Using key features and key frames, they were able to avoid redundant frame processing and executing high cost bundle adjustment in a real-time manner. The focus of PTAM is it's real-time tracking and usage of sparse features result in a sparse point cloud models which make it difficult to use for dense reconstruction. Another state of the art approach is LSD-SLAM [5] which runs on CPU real-time using semi-dense depth maps and it's following work Direct Sparse Odometry[1], which improves accuracy and performance. Our approach was proposed by *Izadi, S. et. al.* [2]. In their work, they leverage the compute power of the modern-day GPGPUs coupled with the commodity depth sensor Kinect to accomplish a real-time dense reconstruction of the scanned surface out of depth images. Kinect uses structured IR light to mea-

sure depth. A global static reconstruction model is used as a reference for each newly observed frame. Correspondences of a new frame are found in the global model, and the reconstruction is fused with the new frame's TSDF values during the Volumetric Integration step.

4 Method

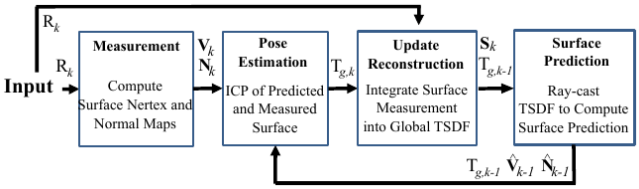

Figure 2: Overall System Workflow [2]

4.1 Surface Mesurement

We followed the same methodology as [2], where at each frame k we get the raw depth map which is with calibrated depth measurements $R_k(u) \in \mathbb{R}$ with u being a pixel in the image domain. We construct an image pyramid, then apply a bilateral filter to R_k at each pyramid level and have a depth map D_k with reduced noise. Afterwards, back-projecting the depth values to the sensor's frame of reference produces the vertex map V_k. We then compute the normal map from the vertex map we created using central differences for every level.

4.2 Surface Reconstruction

4.2.1 Voxel grid representation of scene

We represent the scene as a 512x512x512 voxel grid with 0.01m as our voxel size. We assume that the camera is in the center of the grid in the first frame received from the sensor such that we are able to switch camera and grid coordinate systems easily by just scaling the 3D points we are processing.

In figure 3, a slice of the voxel grid can be seen.

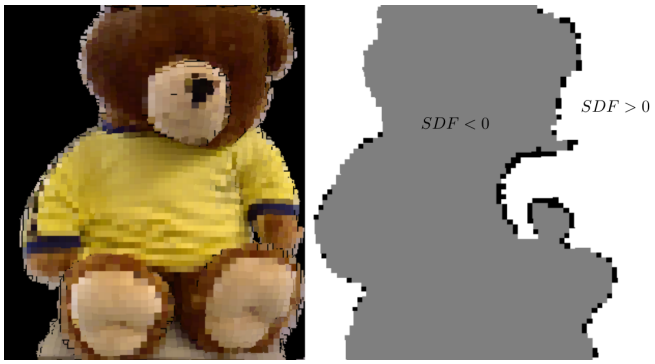

Figure 3: Left: Raycasted surface. Right: Slice of the TSDF. Zero crossings black, negative values grey, positive values white.

4.2.2 Integration of the volume

Every time we receive a new depth frame, we go through every voxel in the grid, project them back to the depth image using the previous camera pose, then update their distance and weight values in a weighted average fashion mentioned in the paper [2]. If the voxel is out of the camera frustum, or it is too far away from the surface, it remains unmodified. The update step for a single voxel is as follows:

$\dot{u}$ = u in homogenous coordinates
$\lambda = ||K^{-1}\dot{u}||$
sdf = depthImage(u) - $\frac{1}{\lambda}||translation - v^g||$
if $sdf >= -truncation$ **then** ▷ if sufficiently close to surface
  if $sdf > 0$ **then**
    newValue = $min(1, sdf/truncation)$
  else
    newValue = $max(-1, sdf/truncation)$
  currTSDF = grid(v).distance
  currWeight = grid(v).weight
  newWeight = 1
  grid(v).distance = $\frac{currTSDF*currWeight+newValue*newWeight}{currWeight+newWeight}$
  grid(v).weight = min(newWeight + currWeight,W_{max})

Figure 4: Application of the signed distance function to a valid voxel in the grid

In this step we have 2 important hyperparameters to decide:

- Truncation value
- Maximum allowed weight

Too low truncation values result in loss of information because it would constrain the updating step too much, whereas too high values result in redundant computation and errors in ray-casting. After testing out with multiple values and sequences, we decided on a value of 5 voxels.

4.3 Surface Prediction

In order to have a dense representation of the scene, we cast rays to the scene from the current estimated camera position. For every pixel, we compute a vector and we shoot a ray from the camera in that direction along the voxel grid until either we encounter a zero-crossing or go out of the grid. One hyperparameter here is the length of one ray step. Initially, we set it to a single voxel edge. Setting something smaller leads to denser and slightly more accurate surface predictions, but also reduced performance.

4.4 Pose Estimation

4.4.1 Correspondences

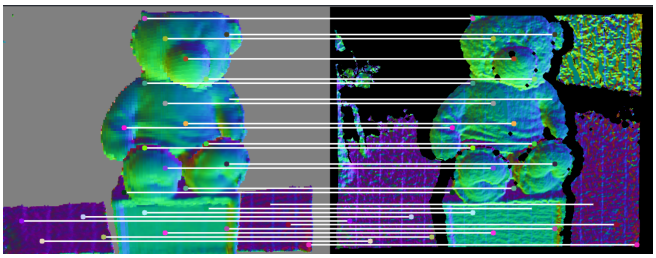

Figure 5: Visualization of some of the correspondences achieved with projective data association. Left: Normal reconstruction from the model. Right: Normal reconstruction from the sensor's depth image.

For finding correspondences we again use the same method suggested in [2], which is the projective data association method. It must be noted that this method heavily relies on the small inter-frame movement assumption. Since the model is built throughout the process and is more robust and less noisy than the raw depth values received from the camera, model-to-frame tracking for ICP is preferred [2].

4.4.2 ICP

During the ICP step, as in [2], we are linearizing it with the small movement assumption. The frame-to-model tracking is realized by aligning the incoming surface measurement (V_k, N_k) with the model prediction from the previous frame (V^g_{k-1}, N^g_{k-1}) using point-to-plane error metric which is as follows. Then the point-plane energy term is as follows:

$$E(T_{g,k}) = \sum_u ||(T_{g,k}V_k(u) - V^g_{k-1})^T N^g_{k-1}(u)||_2$$

And the linearized incremental transform, where z denotes the iteration number, is as follows:

$$T^z_{inc} = \left[R^z|t^z\right] = \begin{bmatrix} 1 & \alpha & -\gamma & t_x \\ -\alpha & 1 & \beta & t_y \\ \gamma & -\beta & 1 & t_z \end{bmatrix}$$

The incremental transform after each iteration, parametrized x vector and the incremental point transfer are then as follows:

$$T^z_{g,k} = T_{inc}T^{z-1}_{g,k}$$

$$x = (\beta, \gamma, \alpha, t_x, t_y, t_z)$$

$$T^z_{g,k}V_k(u) = R^z V^g_k(u) + t^z = G(u)x + V^g_k$$

We construct the full A^TA and the corresponding A^Tb as follows and solve it using singular value decomposition.

$$G(u) = \left[\left[V^g_k(u)\right]_X | I_{3x3}\right]$$

$$A^T = G(u)^T N^z_{k-1}(u)$$

$$b = {N^g_{k-1}}^T(V^g_{k-1} - V^g_k)$$

The above terms are obtained by deriving the energy term with respect to x and setting it to zero as in [2] to minimize the error in the alignment.

$$\min_{x \in \mathcal{R}^6} \sum_u ||{N^g_{k-1}}^T(G(u)x + V^g_k - V^g_{k-1})||^2_2$$

We also tried to use the proposed way in the paper, however, we weren't able to make it work for us. In [2] they sum up the terms in the normal equation in the GPU and solve the 6×6 system in the CPU via Cholesky decomposition. This would have severely improved our performance as our main bottleneck currently is the SVD we are solving. The normal equation mentioned is also as follows using the A and b terms constructed before:

$$\sum_u (A^TA)x = \sum A^Tb$$

5 Experiments and Results

All experiments were performed using *TUM RGB Dataset* [5]. We could not generate our own sequences due to the lack of a Kinect sensor. We have run our code at our own personal desktop PC, which is equipped with a NVIDIA RTX 3070 GPU. And timings of individual methods are as follows:

Step	Time(ms)
Pose Estimation	90
Update Reconstruction	52
Surface Prediction	32
Surface Measurement	26

The correctness of our method was tested in two main scenarios: *freiburg1_xyz* which focuses on translation and *freiburg1_rpy* which mainly tests rotation. These sequences allowed us to find flaws in our implementation during development.

Pose estimation is a core part of the method. An incorrectly estimated pose would end in a wrong TSDF because mistaken voxels would be updated complicating the alignment of further frames. If the small-angle assumption in consecutive movements is not respected in consecutive frames the estimated pose using linearized ICP will fail. This quickly leads to the failure of the application, since it cannot find any correspondences between frames due to wrong updates and predictions. We have also evaluated our method with *freiburg1_360* sequence which includes fast movements between frames and our model was not able to produce a clean reconstruction. An example can be seen in figure 6 where the reconstruction becomes very noisy due to fast movement between frames and incorrect TSDF updates.

Scenes with large planar objects filling most of the sensor's field of view are the main failure case for this method. These types of scenarios produce an unconstrained linear system for the pose estimation because the correspondences are estimated using point-to-plane matching criteria.

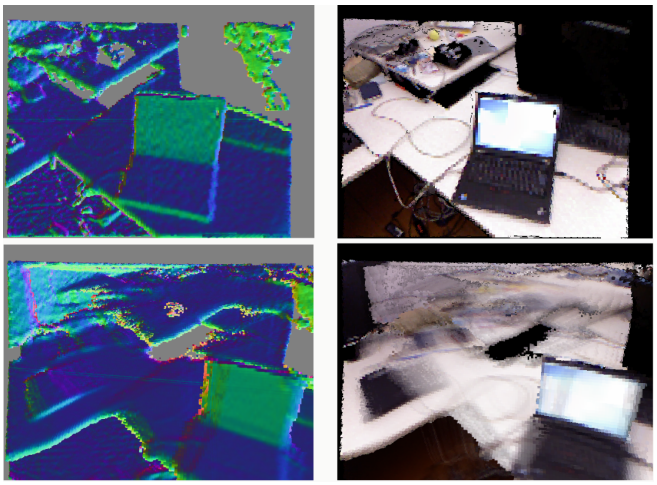

Figure 6: Failure of the reconstruction due to violation of small-angle assumption. Comparison of the reconstructed surfaces at frame 1 and 10.

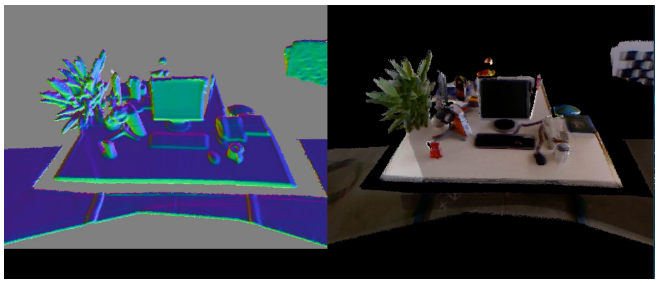

Figure 7: Predicted surface generated from *freiburg1_rpy* dataset. The movement of this sequences is particularly focused on rotations. The method is able to estimate the poses properly using linearized ICP.

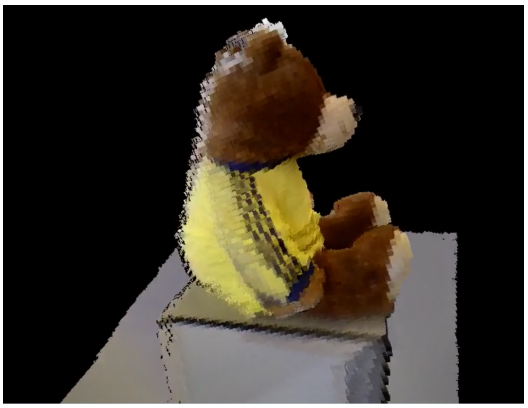

Figure 8: Predicted surface generated from *freiburg1_teddy* dataset. Reconstruction of the dataset after 300 frames.

Due to time and resource limitations we were not able to fully optimize the pipeline for real-time applications, however we are able to run the whole pipeline in around 5 frames per second.

6 Conclusion

In conclusion, our implementation successfully follows the paper's pipeline and is capable of reconstructing dense surface representations given that the small-angle assumption is fulfilled. Our only shortcoming is the lack of real-time performance. We were not able to implement the ICP step same as the paper, which led to a performance bottleneck because we generate a larger system of equations: *(Number of matches) x 6*. And it is costly to solve such a system at every ICP iteration. We have also observed that the method is strongly dependent on the estimated camera position. A limitation of our implementation is solving the linear system to estimate the camera position. Although we implemented the approach described by [2], which allowed us to take advantage of modern GPGPUs, the estimated camera positions were not accurate enough to process long sequences with it. For that reason, we solve a linear system of dimensions $\#correspondences \times 6$ using SVD in CPU. This system has bigger dimensionality than the one suggested by the original work but it produced better results in our experiments.

References

[1] J. Engel, V. Koltun, and D. Cremers. Direct sparse odometry. In *arXiv:1607.02565*, July 2016.

[2] Shahram Izadi, David Kim, Otmar Hilliges, David Molyneaux, Richard Newcombe, Pushmeet Kohli, Jamie Shotton, Steve Hodges, Dustin Freeman, Andrew Davison, and Andrew Fitzgibbon. Kinectfusion: Real-time 3d reconstruction and interaction using a moving depth camera. In *Proceedings of the 24th Annual ACM Symposium on User Interface Software and Technology*, UIST '11, page 559–568, New York, NY, USA, 2011. Association for Computing Machinery.

[3] Georg Klein and David Murray. Parallel tracking and mapping for small AR workspaces. In *Proc. Sixth IEEE and ACM International Symposium on Mixed and Augmented Reality (ISMAR'07)*, Nara, Japan, November 2007.

[4] Georg Klein and David Murray. Parallel tracking and mapping on a camera phone. In *Proc. Eigth IEEE and ACM International Symposium on Mixed and Augmented Reality (ISMAR'09)*, Orlando, October 2009.

[5] 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 International Conference on Intelligent Robot Systems (IROS)*, Oct. 2012.