Object Reconstruction with Depth Error Compensation Using Azure Kinect

Kaiyue Shen
ETH Zurich
kashen@ethz.ch

Yunke Ao
ETH Zurich
yunkao@ethz.ch

Yifei Dong
ETH Zurich
yifdong@ethz.ch

Panayiotis Panayiotou
ETH Zurich
ppanayio@ethz.ch

Abstract

Classical 3D reconstruction methods have the built-in problem of not being able to separate the object of interest from the rest of the scene. Most methods also suffer from shape distortion caused by inaccurate depth measurement or estimation. In this paper, we present a method for creating precise object meshes automatically based on existing SLAM frameworks (ORB-SLAM2 & BADSLAM) alongside a learning-based depth error compensation mechanism for Time-of-Flight cameras. We demonstrate the effectiveness of our method by using the Azure Kinect RGBD camera to reconstruct various objects. We also conduct experiments showing better performance of our depth error compensation mechanism compared with the classical depth correction method adopted by BADSLAM.

1. Introduction

Object reconstruction is the process of capturing the 3D structure and appearance of real objects into a mesh. Meshes encode the 3D structure of objects and can be used for various applications e.g. 3D printing, AR etc. Current object reconstruction methods have the limitation that there is no obvious way to separate an object of interest from the background. We explore this task by scanning a scene with and without the object of interest and then attempting to reconstruct only the object of interest. The described pipeline is composed of five processes: point cloud generation (Panayiotis), background removal (Yifei & Yunke), geometric registration (Yifei & Yunke), surface reconstruction (Kaiyue) and system bias correction (Yifei & Yunke & Kaiyue). We denote in parentheses the responsible group members for each part of the project.

Point cloud generation generates a set of data points with depth and RGB values from a set of images of a static scene. Note that given an RGBD image, one can trivially construct this point cloud. The challenge lies in having multiple images of the same static scene and adding points from all images to the same point cloud keeping a common reference frame. This can be tackled by using a *Simultaneous Localization and Mapping (SLAM)* technique to simultaneously determine the camera pose and build a 3D model of the scene.

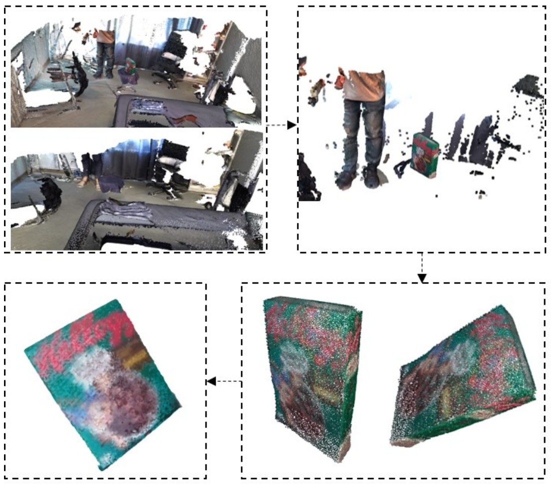
Figure 1: 3D object reconstruction steps for a green box, including point cloud generation (top left), background removal (top right & bottom right), registration and surface reconstruction (bottom left).

In our project, we generate point clouds from multiple scans, one without the object of interest and one or more including the object. The additional scans can be useful in capturing parts of the object which are not initially visible (e.g. the bottom). We combine information from different scans in subsequent steps. Geometric registration is the process of transforming different point clouds into one coordinate system. Background removal is the process of subtracting the background point cloud from the whole scene point cloud and only keeping the point cloud of the object of interest. At first, for each scan including the object, we register it with the point cloud that doesn't include the object. We post-process the registration outliers to extract a point cloud of the object of interest. The next step is to combine the object point clouds extracted from scans with geometric registration into a single object point cloud. Mesh recon-

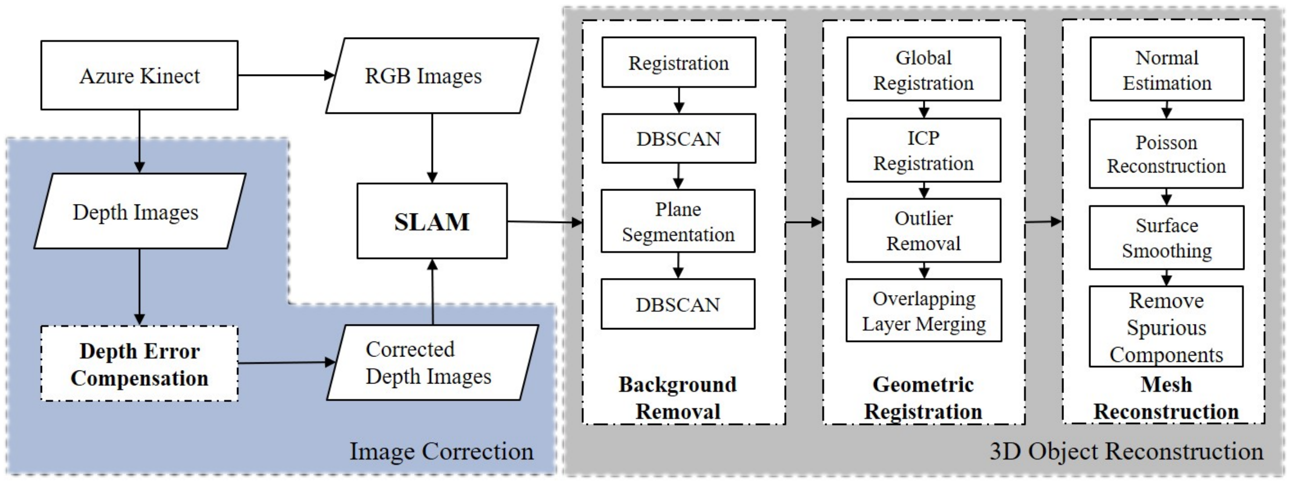

Figure 2: Framework of object reconstruction with depth error compensation using Azure Kinect.

struction finally builds a 3D mesh from the obtained point cloud through classical Poisson reconstruction methods.

System bias correction is the process of compensating the depth error of the depth camera by applying machine learning on a calibration dataset (e.g. Aprilgrid pattern), aiming to obtain a better mesh. The depth error we focus on includes wiggling error and reflectivity dependent errors that are not caused by inaccurate calibration [5].

In conclusion, the five-step pipeline transforms a few videos (at least one video of the object and one scene without it) from an RGBD camera into a corrected 3D object mesh ready for further applications. Our main contributions include:

- We build an automatic object surface reconstruction framework based on existing SLAM frameworks.
- We apply machine learning to compensate depth error of Azure Kinect camera, which outperforms the undistortion method used by BADSLAM in terms of RMSE.

2. Related Work

For point cloud generation, we register depth and color cameras using standard calibration methods for the pinhole camera model as described in *Hartley and Zisserman* [8]. For combining the points into a single coordinate frame we use *SLAM*. *SLAM* is a very well researched algorithm [19] [15] and in this work we merely use some of the SOTA methods (*BADSLAM* [19], *ORB-SLAM2* [15]) comparing their performances on our task. We note the distinction of dense (*BADSLAM*) and feature-based (*ORB-SLAM2*) SLAM methods. In feature-based methods the optimization is done using only a sparse set of keypoints (features) while in dense methods all the points contribute to the error to be minimized which causes a higher runtime and computational cost. More specifically, in *ORB-SLAM2* BA is performed on a sparse set of keypoints, minimizing reprojection error of the reconstructed points while in *BADSLAM* direct BA is performed minimizing a photometric and geometric error.

For background removal, the existing approach applied by KinectFusion is first using geometric registration algorithms to detect points with significant disparity to the background map [11]. Then connected patches are clustered using DBSCAN [4] to extract the target object point cloud. We also adopt this idea in our framework.

For geometric registration, the classical iterative closest point algorithm (ICP) is widely applied [2] [13]. ICP algorithm requires a set of initial rotations and transformations since it only converges to the nearest local minimum of a distance metric. Good initialization of the transformation matrix could be achieved using RANSAC-based global registration [3]. Fast global registration proposed in [21] could further speed up the optimization process. In the case of registration of colored point cloud sets generated by RGBD camera, a colored point cloud registration method is proposed synthesizing geometric and photometric information [16]. All these methods are applied to obtain precise registration in our framework.

Poisson surface reconstruction [10] is a typical mesh reconstruction method. It reconstructs a triangle mesh from a set of oriented 3D points by solving a Poisson system. In order to preserve the color information, one way is to keep the color of point cloud and extrapolate the color values to the vertices of the reconstructed mesh, and another [12] [6] is to reconstruct the mesh first and then project the texture onto faces using the captured RGB-D sequences. Taking the time consumption into account, we choose the first one in our implementation.

BADSLAM includes the non-calibration depth error correction in its intrinsic optimization process [19], where the depth values are corrected according to the model proposed by [9]. However, non-calibration depth error of Tof camera is influenced not only by the direction of light emitted from the camera, but also by the distance and reflection properties of the measured surface [5]. Fuchs and Hirzinger model the distance-related offset as a third-order polynomial [7]. Ferstl et al. proposed a Random Forest (RF) to predict the depth error using depth value and pixel position together with intensity features [5]. This approach is most suitable for our purpose to include more features for depth error compensation of *BADSLAM*. In our pipeline, we use both RF and Neural Network (NN) to train models to correct the depth offset error given intensity features and geometry features.

3. Point cloud generation and SLAM

The input to point cloud generation is an RGBD video of a static scene and the output is a combined 3D RGB point cloud of all the frames. To start with, we use the *Azure Kinect* as our camera. The *Azure Kinect* has 2 cameras, a depth camera and a color camera, which are separate devices capturing from different points of view and have different resolutions. As a result, one does not get an RGBD video directly, but rather an RGB video and a separate depth video. Neither the resolution, nor the camera pose nor calibration nor the frame timing match between the two camera feeds. We first want to perform RGB and Depth registration to align the RGB and Depth frames and then use them as part of a SLAM pipeline.

We implement registration which takes video frames (RGB and Depth) and registers them using the extrinsic and intrinsic characteristics of the two cameras. We need the intrinsic matrices $\mathbf{K_{rgb}}$, $\mathbf{K_{depth}}$ for both cameras and a transformation matrix from one camera to the other $\mathbf{P}$ (which can be computed from the two extrinsic matrices, but in the *Azure Kinect* is provided with an API call).

$$\mathbf{P} = \left[\begin{array}{c|c} \mathbf{R} & \mathbf{t} \\ \hline \mathbf{0} & 1 \end{array}\right] \tag{1}$$

$$\mathbf{K_{depth}} = \begin{pmatrix} fx_d & s_d & cx_d \\ 0 & fy_d & cy_d \\ 0 & 0 & 1 \end{pmatrix} \tag{2}$$

$$\mathbf{K_{rgb}} = \begin{pmatrix} fx_{rgb} & s_{rgb} & cx_{rgb} \\ 0 & fy_{rgb} & cy_{rgb} \\ 0 & 0 & 1 \end{pmatrix} \tag{3}$$

In *Algorithm 1* (largely inspired from [14]) we describe how to map depth pixels to color pixels, getting 3D points with RGB values.

After getting registered RGB and Depth video tracks we want to use an RGBD version of *SLAM* on top of them to build a map of the scene. SLAM is an algorithm for simultaneously constructing a 3D model of a scene and keeping track of the pose & location of a camera in it. After performing *SLAM* we have a camera pose and trajectory which we can use to align the point clouds of each frame (transform all of them to the same coordinate frame). There are various algorithms for achieving this but here we consider *RGB-D SLAM* SOTA approaches *BADSLAM* and *ORB-SLAM2*.

Algorithm 1: RGB and Depth Registration

```
Data: $\mathbf{K_{rgb}}, \mathbf{K_{depth}}, \mathbf{P}$
1. Map each pixel $(x_d, y_d)$ to 3D world space $W$.
   $W.x = (x_d - cx_d) \times depth(x_d, y_d)/fx_d$
   $W.y = (y_d - cy_d) \times depth(x_d, y_d)/fy_d$
   $W.z = depth(x_d, y_d)$
2. Transform to the coordinate frame of the RGB
   camera $C = \mathbf{R} \times W + \mathbf{t}$
3. Project each of the 3D points to the color frame
   $Cframe.x = (C.x \cdot fx_{rgb}/C.z) + cx_{rgb}$
   $Cframe.y = (C.y \cdot fy_{rgb}/C.z) + cy_{rgb}$
4. From the x,y values in the color frame we can
   reject points that are outside the frame and keep
   the RGB value for each point that is within the
   frame. This way we just created an $\{x, y, z, r, g, b\}$
   tuple i.e. an aligned frame.
   $C.rgb = rgb(Cframe.x, Cframe.y)$
Result: Aligned frames $C(x, y, z, r, g, b)$
```

We notice that both choices perform similarly and have similar issues. More importantly they very easily lose track if the camera is moving “too fast”. While we notice that *ORB-SLAM2* loses track less easily than *BADSLAM* we choose to use *BADSLAM* for our practical experiments. *BADSLAM* has a live-feature implemented and can perform *SLAM* with live input from the *Azure Kinect* camera. In practical applications, such as ours, this is a huge advantage since it can notify the user that they’re moving too fast (or any other problem) while they are still recording. We further note that *BADSLAM* does have implemented point cloud generation and RGB & Depth registration but we still implemented these to be able to compare with *ORB-SLAM2*. We provide our code for both methods along with instructions for usage.

4. Surface reconstruction

4.1. Background removal

The pipeline of background removal is illustrated in figure 2. The inputs are two point clouds of a scene. One of them contains the target object and the other does not. The output is a point cloud of the target object.

We first register two input point clouds and obtain the outliers. Afterward it is cropped to get a smaller area con-

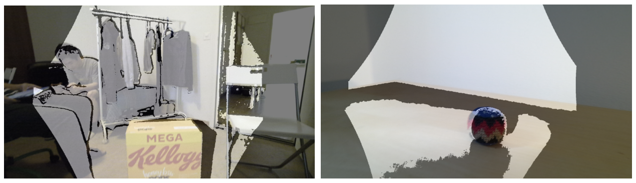
Figure 3: Demonstration of RGB and Depth registration by overlaying aligned color and depth frames. While this works effectively in some scenarios (left), some parts of the scene could cause trouble e.g. reflections on the table (right). The rhombus shape indicates the difference in resolutions between the two cameras.

taining points of the target object. The crop operation follows an approximate guess of distance from the object to the camera. The density-based clustering algorithm, DBSCAN [4], is then implemented to extract the object point cloud. As an optional choice, a plane could be segmented out from selected points using RANSAC before clustering to remove the points of the surface that the object is placed on. We select the target object from the clusters following several criteria: (1) Minimum outlier ratio is 0.95; (2) The interval of number of points in the cluster is $[1000, 40000]$.

4.2. Geometric registration

In the previous steps, our algorithm generates two or more 3D point cloud sets of a target object. The object is placed in different poses in the world coordinate system in different sets so that the object surface area could be captured completely. Additional scans can be useful to capture parts of the object which were not visible in one scan by changing its position in the scene (e.g. the bottom part of the object). The inputs of geometric registration are two or more point clouds with arbitrary initialization transformation. The output is a transformation matrix that perfectly aligns the point clouds.

Our algorithm synthesizes the advantages of several registration methods to optimize the running time. It is demonstrated in the geometric registration section of figure 2. Instead of taking a set of arbitrary transformations as initialization, which suffers from the risk of ICP failure, we first generate a rough alignment through RANSAC-based global registration. At the end, outliers are removed to suppress the noise and overlapping layers are merged.

In global registration, a Fast Point Feature Histograms (FPFH) feature is calculated for each point after downsampling the point cloud. The FPFH feature is a vector composed of 33 elements containing information of 3D coordinate and estimated surface normal [18]. Random sample consensus (RANSAC) is then applied to find a rough alignment of point clouds with a nearest-neighbor querying method in feature space.

Iterative closest point algorithm (ICP) provides a refined alignment for two point clouds. In each iteration, a corresponding set $K = \{(p, q)\}$ is built from source point cloud P and target point cloud Q transformed with current transformation matrix T. Then T is updated by minimizing the objective of a point-to-plane distance metric [17]:

$$E(T) = \sum_{(p,q)\in K} ((p - Tq) \cdot n_p)^2 \tag{4}$$

where n_p is the normal of point p.

4.3. Mesh reconstruction

Mesh reconstruction is to reconstruct a smoothed colored mesh from the registered point cloud data. It can mainly be separated into two steps as shown in figure 2: (1) Mesh reconstruction; (2) Mesh cropping.

Since our registered point clouds are not oriented, we first estimate the normal for points using standard moving least-squares method [1]. For each point in the cloud, it picks the nearest neighboring points and finds a local plane that minimizes the sum of square distances. Next, we perform the Poisson Surface Reconstruction [10], which reconstructs a triangle mesh from a set of oriented 3D points by solving a Poisson system. The idea is that the gradient of the indicator function χ should be equal to the surface normal field $\vec{V}$ when near the surface and zero otherwise, i.e. $\min_\chi \|\nabla\chi - \vec{V}\|$. Since we find some reconstruction results especially those unclosed meshes have redundant parts (see in Appendix), we then compute the oriented bounding-box containing the raw point cloud, and use it to filter all surfaces from the mesh outside the box. Finally, after performing the smoothing, we find some small meshes exist due to noise. So we implement a connected components algorithm to cluster the triangles and remove components with triangle numbers less than 500.

5. Depth Error Compensation

The goal of depth error compensation is to reduce the offset of depth images before feeding them to the SLAM and reconstruction pipeline. In the original framework of BADSLAM, each pair of raw RGB and depth images will first be aligned and reshaped to the same size. Then a shared camera intrinsic will be optimized for both the RGB and depth image. These preprocessed images and intrinsic are input to the following odometry and mapping framework. In our modified pipeline, each pair of preprocessed images and intrinsic will first go through our depth error correction pipeline before the following process.

For each input pairs of depth and color images, we will predict the depth error at each pixel of the depth image using a learned model. The extracted features at each pixel include $f(u, v) = \{u, v, d, \frac{\partial d}{\partial u}, \frac{\partial d}{\partial v}\, I, \frac{\partial I}{\partial u}, \frac{\partial I}{\partial v}, \frac{\partial^2 I}{\partial^2 v} + \frac{\partial^2 I}{\partial^2 u}\}$, where u, v are pixel positions, d is the depth value at each

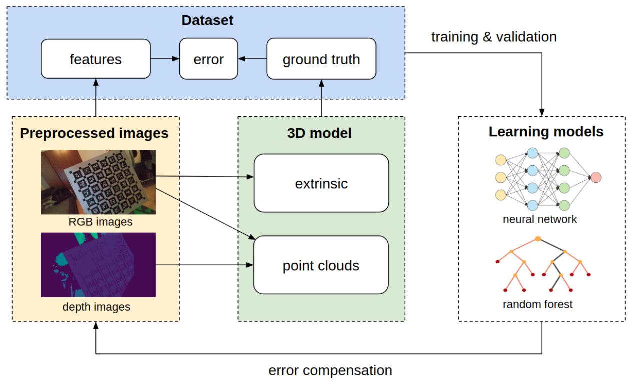

Figure 4: Framework for depth error compensation

depth image pixel, I is the intensity values at each color image pixel. To train the model, we collect training and validation images by scanning an Aprilgrid target board from different directions. Given the calibration target, we first generate the ground truth d_t of each depth pixel, then different models are used to learn the mapping from extracted features $f(u, v)$ to the depth error $d_t - d$. The framework of depth error compensation is shown in figure 4.

5.1. Training Data Generation

Before calculating the gradient features of images, we first apply the bilateral filter to smooth the depth image. The first-order gradients of images are obtained by the Sobel filter, and the second-order features are extracted using Laplacian filter. All the features are scaled to range from -1 to 1 except for the measured depth, which is scaled by a known depth ratio r from raw pixel value to real depth.

For ground truth generation, our pipeline is inspired by [5]. Firstly, we extract the apriltags from color images using AprilTag library [20]. Given centers of the apriltags as target points, the extrinsic T_{wc} between the camera and the target board can be determined by solving the Perspective-n-Point (PnP) problem. Then the measured point cloud $\{P_w^i = [x_w^i, y_w^i, z_w^i]^T\}_{i=0}^n$ is projected to the world coordinate according to the camera intrinsic K and depth d^i:

$$P_w^i = T_{wc} \cdot K^{-1} \frac{d^i}{r} \begin{bmatrix} u^i \\ v^i \\ 1 \end{bmatrix} \tag{5}$$

Only a part of the point cloud inside a convex hull $\{P_w | 0 < x_w < a, 0 < y_w < b, |z_w| < thr\}$ is selected to generate training data, where the corner target of the target board is set as the origin of the world coordinate, XY plane is the target board plane ($s_w = [0, 0, 1, 0]$), a, b are width and height of the board respectively, and thr is a small value that guarantees to only select the point cloud of the board.

To obtain the ground truth, we first transform the Aprilgrid board plane to the camera frame: $s_c = T_{wc}^{-T} s_w$. Then we define the ground truth positions P_{ct}^i of each measured target board point in the camera frame as the intersection of the viewing ray of the corresponding depth pixel with the true target board plane s_c. The ground truth point coordinates P_{ct}^i and depth z_{ct}^i satisfy:

$$P_{ct}^i = z_{ct}^i K^{-1} \begin{bmatrix} u_i \\ v_i \\ 1 \end{bmatrix} = z_{ct}^i \overline{P}_{ct}^i \tag{6}$$

where $\overline{P}_{ct}^i$ denotes the normalized coordinates of ground truth points. So the true depth value d_t^i is calculated by:

$$n_c \cdot P_{ct}^i + a_c = 0 \tag{7}$$

$$\Rightarrow \quad z_{ct}^i = -\frac{a_c}{n_c \cdot \overline{P}_{ct}^i} \tag{8}$$

$$\Rightarrow \quad d_t^i = r \cdot z_{ct}^i = -\frac{r a_c}{n_c \cdot \overline{P}_{ct}^i} \tag{9}$$

where n_c is the normal of the target board plane (first three elements of s_c), a_c is the last element of s_c.

5.2. Learning Depth Error Compensation

We use both neural network and Random Forest to train models to learn the mapping from input features to depth error. Fully connected deep neural networks are suitable for modeling complex nonlinear mappings. In our problem, we build a network with 2 fully connected hidden layers, as with more hidden layers the model seems to overfit to the training dataset. The dimensions of layers are 1024 and 256 respectively. To avoid overfitting, l1-l2 regularizations and dropouts are added to each layer. Using an Adam optimizer, the model is trained with a learning rate decay schedule.

Random Forest is a meta estimator with an ensemble of decision trees. Each decision tree only learns a subset of the dataset and the meta estimator uses the average of decision trees to improve accuracy and control overfitting []. For RF regression, we use 16 decision trees with maximum depths of 16. The quality of splits of decision trees is evaluated by mean squared error.

6. Experiment

6.1. Surface Reconstruction

As stated in Sec. 3, we use both *ORB-SLAM2* and *BAD-SLAM* and find they have similar performance. So we simply show the result using use *BADSLAM* in the following experiment part. First, we perform *BADSLAM* twice to get sets of point clouds with and without objects using *Azure Kinect* camera. In the background removal stage, we crop the outliers with $2 \times 2 \times 2$ box given initial distance between camera and object, and set density parameter ϵ in DBSCAN to be $0.35 \times voxelSize$, where $voxelSize$ equals 0.05.

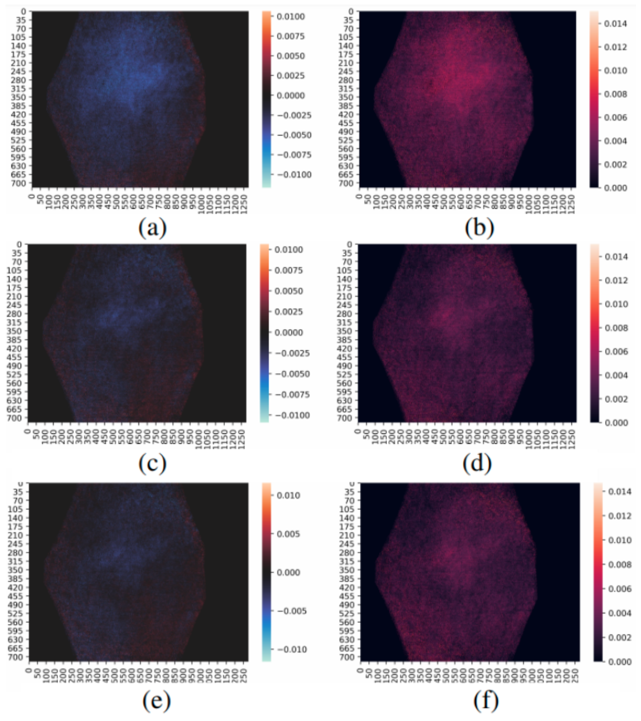
Figure 5: Mean (left column) and standard deviation (right column) of depth error at each pixel among 100 original images (a)-(b) and corrected images using neural network (c)-(d) and random forest (e)-(f).

In Geometric registration, both global registration and ICP registration choose the voxel size as 0.004. In the mesh reconstruction, the maximum octree depth of the reconstructor is set to 8 in order to keep a balance between reconstruction speed and quality. We directly use the color of point cloud instead of doing texture mapping since we find the latter method takes minutes, which is far from real-time. Generally, each step above takes less than 10 seconds.

We show one example of surface reconstruction stage outputs in Figure 1. We note that after the first registration, most area of the background other than the green box is removed. A relatively noiseless point cloud is obtained after the clustering. After implementing geometry registration and mesh reconstruction, we obtain an aligned point cloud with most outliers removed. A smoothed colored mesh could be generated in the end.

6.2. Depth error compensation

We collect preprocessed depth and color images of an Aprilgrid board with Azure Kinect RGBD camera. The size of the board is $80cm \times 80cm$, where the side length and interval of the April tags are $8.8cm$ and $2.64cm$ respectively. We take 1000 images of the Aprilgrid board with 15Hz. The size of aligned depth and color images is 1280×720. To

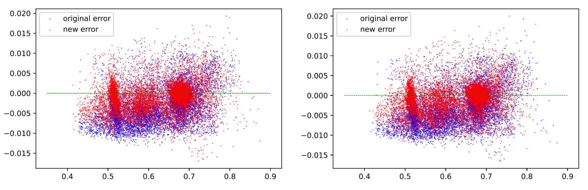
(a) Neural network (b) Random forest

Figure 6: Increment of error w.r.t. measured depth before and after error compensation.

	RMSE (m)	Std (m)	Mean (m)
Original	0.00534	0.00432	-0.00314
BADSLAM	0.00433	0.00428	0.00064
Neural Network	0.00403	0.00382	-0.00127
Random Forest	0.00405	0.00387	-0.00119

Table 1: Comparison of RMSE, standard deviation (Std) and mean of depth error among original image, BADSLAM correction, neural network and random forest correction.

obtain a reasonable size of training data, we only select one from five images to generate data. For each selected image, we augment the dataset by flipping the image vertically, horizontally, and both together. We only select 5% of the points in the convex hull to extract training data. The same generated data are fed to train both the random forest and neural network model for comparison. Validation images are selected from the remaining images in the original dataset.

The results of depth error compensation are shown in figure 5, 6 and table 1. Furthermore, we calculate the mean error and standard deviation error of 100 target board images at each pixel, which is shown in figure 5. The results indicate that our algorithm universally reduces the depth error, given any input image. This is also numerically proved in table 1, with the mean of depth errors decreased by 62.1% after random forest training. The data in table 1 implies the error reduction effects of the neural network algorithm are comparable with that of random forest algorithm, showing better performance than the methods adopt by BADSLAM. Figure 6 demonstrates the error accumulates as depth increases and our approaches substantially reduce the error.

7. Conclusion

We presented a method for creating precise surface meshes automatically from objects and a depth error compensation mechanism. This allows to create 3D object meshes from a few videos of an RGBD camera. As future work, one might consider further extending the pipeline to a real-time interactive object surface reconstruction framework with depth error compensation.

References

[1] J. Berkmann and T. Caelli. Computation of surface geometry and segmentation using covariance techniques. In *IEEE Transactions on Pattern Analysis and Machine Intelligence*, volume 16, pages 1114–1116. IEEE, 1994. 4

[2] P. Besl and N. Mckay. A method for registration of 3-d shapes, ieee trans. *P flattern Anal. and M ac h ine I ntell*, 1(4):23, 1992. 2

[3] R. Cupec, E. K. Nyarko, A. Kitanov, and I. Petrovic. Ransac-based stereo image registration with geometrically constrained hypothesis generation. *Automatika Journal for Control Measurement Electronics Computing & Communications*, 50(3-4):195–204, 2009. 2

[4] M. Ester, H.-P. Kriegel, J. Sander, X. Xu, et al. A density-based algorithm for discovering clusters in large spatial databases with noise. In *Kdd*, volume 96, pages 226–231, 1996. 2, 4

[5] D. Ferstl, C. Reinbacher, G. Riegler, M. Rüther, and H. Bischof. Learning depth calibration of time-of-flight cameras. In *BMVC*, pages 102–1, 2015. 2, 3, 5

[6] Y. Fu. Texture mapping for 3d reconstruction with rgb-d sensor. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 4645–4653, 2018. 2

[7] S. Fuchs and G. Hirzinger. Extrinsic and depth calibration of tof-cameras. In *2008 IEEE Conference on Computer Vision and Pattern Recognition*, pages 1–6. IEEE, 2008. 3

[8] R. I. Hartley and A. Zisserman. *Multiple View Geometry in Computer Vision*. Cambridge University Press, ISBN: 0521540518, second edition, 2004. 2

[9] D. Herrera, J. Kannala, and J. Heikkilä. Joint depth and color camera calibration with distortion correction. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 34(10):2058–2064, 2012. 3

[10] H. Hoppe. Poisson surface reconstruction and its applications. In *Proceedings of the 2008 ACM symposium on Solid and physical modeling*, volume 10, 2008. 2, 4

[11] S. Izadi, D. Kim, O. Hilliges, D. Molyneaux, R. Newcombe, P. Kohli, J. Shotton, S. Hodges, D. Freeman, A. Davison, et al. 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*, pages 559–568, 2011. 2

[12] V. Lempitsky and D. Ivanov. Seamless mosaicing of image-based texture maps. *IEEE Conference on Computer Vision and Pattern Recognition*, pages 1–6, 2007. 2

[13] K.-L. Low. Linear least-squares optimization for point-to-plane icp surface registration. *Chapel Hill, University of North Carolina*, 4(10):1–3, 2004. 2

[14] M. Maghoumi. Align depth and color frames – depth and rgb registration. https://www.codefull.net/2016/03/align-depth-and-color-frames-depth-and-rgb-registration/. 3

[15] 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*, 33(5):1255–1262, 2017. 2

[16] J. Park, Q.-Y. Zhou, and V. Koltun. Colored point cloud registration revisited. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 143–152, 2017. 2

[17] S. Rusinkiewicz and M. Levoy. Efficient variants of the icp algorithm. In *Proceedings third international conference on 3-D digital imaging and modeling*, pages 145–152. IEEE, 2001. 4

[18] R. B. Rusu, N. Blodow, and M. Beetz. Fast point feature histograms (fpfh) for 3d registration. In *2009 IEEE international conference on robotics and automation*, pages 3212–3217. IEEE, 2009. 4

[19] T. Schops, T. Sattler, and M. Pollefeys. Bad slam: Bundle adjusted direct rgb-d slam. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 134–144, 2019. 2, 3

[20] J. Wang and E. Olson. AprilTag 2: Efficient and robust fiducial detection. In *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, October 2016. 5

[21] Q.-Y. Zhou, J. Park, and V. Koltun. Fast global registration. In *European Conference on Computer Vision*, pages 766–782. Springer, 2016. 2