An Active Patch Model for Real World Appearance Reconstruction

Farhad Bazyari(✉) and Yorgos Tzimiropoulos

Department of Computer Science, University of Lincoln, Lincoln LN6 7TS, UK
{fbazyari,gtzimiropoulos}@lincoln.ac.uk

Abstract. Dense mapping has been a very active field of research in recent years, promising various new application in computer vision, computer graphics, robotics, etc. Most of the work done on dense mapping use low-level features, such as occupancy grid, with some very recent work using high-level features, such as objects. In our work we use an active patch model to learn the prominent, primitive shapes commonly found in indoor environments. This model is then fitted to coming data to reconstruct the 3D scene. We use Gauss-Newton method to jointly optimize for appearance reconstruction error and geometric transformation differences. Finally we compare our results with Kinect Fusion [6].

Keywords: Dense mapping · Deformable patches · Gauss-Newton optimization

1 Introduction

Structure from motion and mapping has been studied intensively for a long time in computer vision society both in terms of theory and application. But scarcity of computational power in older systems meant that most systems had to work with sparse, feature-point based maps. However with recent advancements in processing units (both CPU and GPU) and easy access to commodity depth sensor such as Microsoft Kinect, we have seen some breakthroughs in dense mapping systems which in return have paved the path for many new applications in robotics and manipulation, computer graphics and augmented reality and so on which was simply not possible using traditional sparse representations.

Different representations are available for 3D maps such as 3D mesh, which has the appeal that can be readily used by rendering mechanisms (e.g. OpenGL). However there has been growing interest in using non-parametric representations, such as Truncated Sign Distance Function (TSDF) [1] due to their constant complexity which is independent from the complexity of the scene/map they are representing. This characteristic is particularly appealing in on-line applications where fixed complexity means constant operation time and more robustness to motion, etc.

TSDF has been embraced by most dense mapping systems and very promising results have been demonstrated [6,7]. However it is a very low-level representation, storing information at the voxel level only (equivalent to pixels in 2D

L. Agapito et al. (Eds.): ECCV 2014 Workshops, Part I, LNCS 8925, pp. 443–456, 2015.
DOI: 10.1007/978-3-319-16178-5_31

images). On the other hand, there has been a few very recent works that try to make use of higher-level information that might be available to the system such as type of objects to expect in the map [8]. This type of information is problem specific and is only useful in very controlled environments.

In this work we propose the use of mid-level features which while providing better quality maps and more compact representation, are very common in almost any man-made environment and hence are not problem specific. We use deformable 3D patches to learn the most common primitive shapes from training dataset. Loosely speaking these primitive shapes are equivalent to small planar surfaces, sharp corners, etc. and our generative model will later on be able to produce any random 3D map from these building blocks. After model is trained, it can be fitted to new test data to produce a higher quality map. We optimize for appearance and geometric transformation (3D rigid-body transformation in our case) simultaneously. This separation of appearance and geometric transformation will allow for a much simpler model (in terms of number of components) and leads to better results. Optimization is based on Gauss-Newton method and is done iteratively. Finally we make comparison between our results and one of the state-of-the-art dense mapping systems.

2 Related Work

While SLAM systems have been around for more than a decade now [2,4,5], it has been only recently that true dense mapping has been considered in SLAM systems mostly due to the availability of computational power (especially GP-GPUs) and also because of advancements and availability of commodity depth sensors such as Microsoft Kinect.

Kinect Fusion [6] is one of the first practical systems that uses Kinect sensor and returns dense map in form of a TSDF structure. It is a full SLAM system closing the loop between tracking and mapping while sensor pose tracking is done using Iterative Closest Point algorithm [11].

In order to overcome the limitations that active sensors impose, Newcombe at al. introduced DTAM [7] which is in many ways similar to KinectFusion, but it uses a single monocular camera. Hence it can work in much wider scale range and can scan objects much farther away from sensor. Also passive sensing means it has smaller energy requirements and can be used with a wide range of sensors.

In a recent work, SLAM++ [8] have put the object detection directly inside tracking and mapping loop. This will increase the map quality, because noise free model is already available. Also it helps the tracker to track against the correct model hence improving stability. It also provide the possibility of presenting the map in a much more compact form. The problem with this method is that it only works in very controlled environment where we have very good knowledge about the type of objects that can be found in there.

In their work on face pose detection, Tzimiropoulos et al. [9,10] model the variance in different faces in 2D images in terms of shape and appearance variations. Then they train their linear model on real world samples and take an iterative optimization approach for fitting their model to new test data. While quite different in application, their optimization technique is very similar to the one we use in this work.

Probably the closest work to ours is of Zhu et al. [12] who try to learn small 2D patches from a set of high resolution images. These trained patches are later on used to increase the quality of low resolution test images. They jointly train their model for high resolution/low resolution patch dictionary and also for deformation of their patches at fitting stage, to minimize the reconstruction error while making best use of prior knowledge.

Finally we like to mention the related work of Hejrati and Ramanan [3] who focus on object recognition and 3D reconstruction at the same time. They use a simple part based model to reconstruct the object in a way that best agrees with visual evidence while also trying to do recognition and making use of prior information.

3 Method

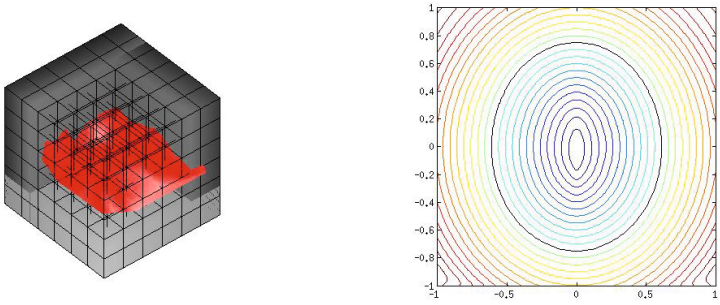
In this section we will first briefly mention data preparation step, then we explain our model and also how Gauss-Newton optimization is used to fit model to (noisy) data.

3.1 Preparing Data

We use TSDF (Truncated SDF) to store and work with 3D structures (this is also used in Kinect Fusion and many other mapping systems). TSDF in its most basic form, divides the working volume into a regular grid of same size voxels. Each voxel stores two numbers ; *value* that indicates shortest distance from center of the voxel to closest surface, and *weight* which is an indicator of our certainty in *value*. By convention; a positive *value* means the voxel is in front of surface (in empty space outside objects) and a negative *value* indicates voxel being located behind a surface (or inside watertight objects) and hence being occluded. True surface is then extracted by (bilinear) interpolation and is marked as zero-set of this 3D scalar volume. The confidence measure (weights) are used in this interpolation step for extracting the zero set.

Figure 1 illustrates SDF in 2D and 3D, in this paper we will visualize only the zero-level of TSDFs and patches which correspond to surfaces in real world. However it is important to remember that our patches are cubes in 3D and when running optimization/fitting, we are dealing with all the information stored in 3D patches and not only information on the zero-level set.

The aforementioned patches are small parts of original TSDF structure in the form of n-by-n-by-n grid. Each patch is in fact a miniature TSDF structure itself and we hope to form a vocabulary of these patches that captures commonly occurring primitive 3D shapes. Figure 2.



(a:3D patch and zero level surface) (b:2D patch with values)

Fig. 1. Signed Distance Function is positive outside surface and negative behind surfaces (inside object) with values linearly proportional to distance to closest surface

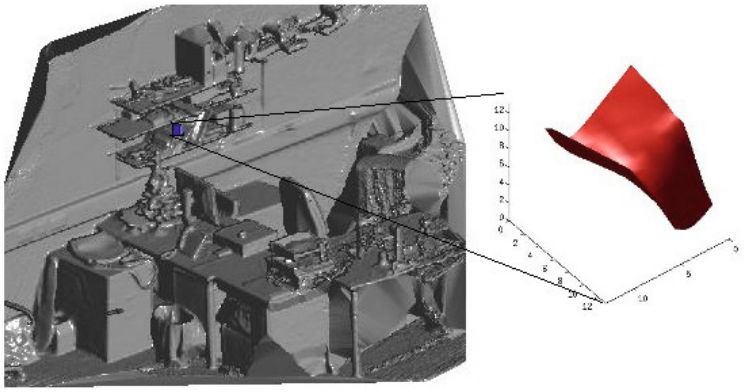


Fig. 2. On left we see extracted surfaces (zero-level set of TSDF structure). And on right, one sample *patch* ($12 \times 12 \times 12$ voxels) is enlarged

3.2 Formation of Patches

In training phase, we have a number of TSDF structures which represent random but noise free data. These data are gathered from indoor environments. In order to be able to recognize the pattern in these data, each of these TSDF structures have to be broken down into smaller patches. Also in the fitting phase, new data comes to us in the shape of a large point cloud (e.g. from a depth sensor) and again we need to break this down into patches that we can work on. So patch formation has to be done both for training and fitting.

Let us consider that we already have a large TSDF structure. We also have a sensor pose for that TSDF. So by performing ray-tracing we can form a large point cloud that sit on the zero-level set (figure 3). (If we were in fitting phase, incoming data was already in form of point cloud and this step was unnecessary,

but the rest applies unchanged.) We are only interested in patches that fall on the surfaces. so a random point in this point cloud is chosen and a partition around this point is taken as a candidate for the patch (red wire-frame cube in figure 3a). Then all these points are concatenated and using Singular Value Decomposition 3 momentum of this cloud is found.

$$[U, S, V] = SVD(\text{point cloud}) \tag{1}$$

momentums are stored as columns of V. Next, point cloud is rotated around its center so that these momentums are aligned with x-y-z axis (optionally we rotate them by $\pi/4$ radian around z axis so that dominant momentums are diagonal. This only makes visualization and code easier) figure 3c. Aligning patches is important because one can imagine that many instances of same 3D structure can occur with different poses. Each of these segments are similar to each other up to a rigid body transformation. By aligning their momentums with x-y-z axis, dependency on point of view is largely eliminated. Now new, and modified, patch is formed around this new point cloud. to do so we form a delaunay triangulation on the point cloud and then perform ray-tracing along the axes with weakest momentum (z-axes in figure 3d), filling the patch while doing so.

3.3 Training the Model and Fitting Data

Training. Our aim is to form a small set of primitive 3D shapes (or eigen-shapes) which can later be combined to give a good estimated reconstruction of any arbitrary scene. For that, we take K random patches from our dataset. Each of these patches consist of n^3 smaller voxels (from underlying TSDF structure). By concatenating these voxels each patch will be deformed into an array of size n^3. If we assume our sample pool is large enough to contain all major primitive structural shape, we can use Principal Component Analysis (PCA) to find a lower dimensional representation for our sample pool.

In order to have a representative set of samples, we can either apply PCA on a very large number of patches, or we can separate the variation in our data to shape variations and appearance variations. Latter approach will save us a lot of computational expense as well as giving a far more compact set. Here appearance is referred to the readings, or *values*, from TSDF while shape variations refer to rigid body transformation. We plan to study the affect of scale variation as well as affine transformation in future work.

So by applying PCA on the appearance space, we can find A_0 which is the average patch, and also we have m shapes (m much smaller than number of samples and also size of sample n^3) that linearly form the m-dimensional appearance space. Matrix A is formed by concatenating these eigen-shapes in its rows. Vector c controls the linear combination of eigen-shapes.

$$\hat{Y} = A_0 + Ac, \quad c = A^T(Y - A_0) \tag{2}$$

Where Y is the patch we are trying to fit the model to and $\hat{Y}$ is reconstructed patch.

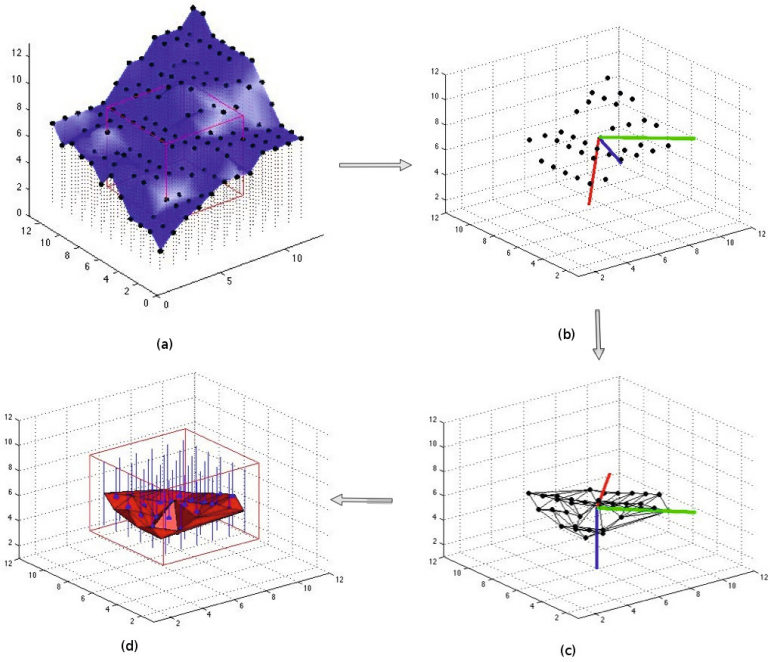


Fig. 3. Starting from top-left and moving clock-wise; (a) TSDF structure that we want to take a sample patch from. Point cloud is formed by ray-tracing. (b) Points that fall into patch are separated and their momentum calculated using SVD. (c) The momentums are aligned with x-y-z axis and delaunay triangulation is formed on them. (d) New, modified, patch is formed by interpolation and ray-tracing.

In the same way, we can linearize the shape space. As mentioned before, in this work we only deal with rigid-body transformation in 3D space. So mean shape, s_0 is a zero vector and 6 eigen-vectors that form matrix S span the 6 dimensional space of rotation and translation in 3D.

$$\hat{s} = s_0 + Sp, \quad p = S^T(s - s_0) \tag{3}$$

More details about this transformation is given in section 3.4.

Fitting. Now having all above, when the new reading arrives, we try to minimize the reconstruction error by solving for both appearance and shape simultaneously.

$$\arg\min_{p,c} ||Y(S(x;p)) - A_0 - Ac||^2. \tag{4}$$

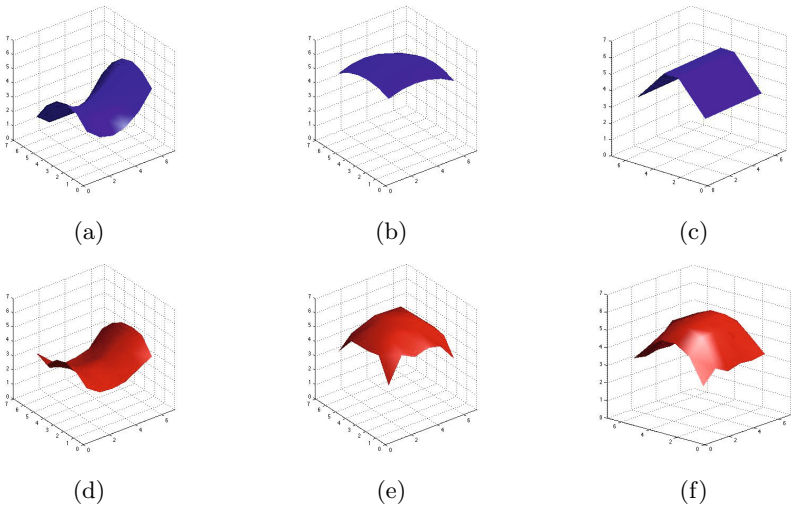


Fig. 4. To illustrate that trained model is capable of reproducing some random and common shapes in 3D, we gave our system 3 random shapes (top row) and model has reproduced the (bottom row).

Where $S(x;p)$ governs the 3D transformation of patch-cube as decided by parameters p, and $Y(S(x;p))$ is the values read from TSDF structure associated to this patch.

from 2 and 3 and by replacing them in 4 we get

$$\arg \min_{\Delta p, \Delta c} ||Y - A_0 + J_0 \Delta p - \sum_{i=1}^{m} (c_i + \Delta c_i)(A_i + J_i \Delta p)||^2. \tag{5}$$

Where J_i is the $N \times n$ Jacobian built as follows: $[A_{i,x}(k) \;\; A_{i,y}(k) \;\; A_{i,y}(k)] \frac{\delta S(x_k;p)}{\delta p}$. And $A_{i,x}(k)$ and $A_{i,y}(k)$ and $A_{i,z}(k)$ are the x and y and z gradients of A_i. ($\frac{\delta S}{\delta p}$ is explained in detail in section 3.4). All above are defined in the model coordinate system, hence $p_0 = 0$. with some abuse of notation we set $J_i = [A_{i,x} \;\; A_{i,y} \;\; A_{i,z}] \frac{\delta S}{\delta p}$.

By omitting the higher order terms from 5 we get:

$$\arg \min_{\Delta p, \Delta c} ||Y - A_0 - Ac - A\Delta c - J\Delta p||^2. \tag{6}$$

Now we can solve for appearance and shape parameters simultaneously, using Gauss-Newton method;

$$[\Delta p; \Delta c] = H_{final}^{-1} J_{final}^T (Y - A_0 - A_c) \tag{7}$$

where $J_{final} = [A; J] \in R^{N \times (m+n)}$ and $H_{final} = J_{final}^T J_{final}$ are final Jacobian and Hessian respectively.

Please note that shape parameters p are used in taking the sample from incoming data, and the dependency of sample Y on p is data dependent and non-linear, hence the step above is done iteratively along with taking new samples and calculating jacobians at each iteration.

3.4 Rigid Body Transformation Formulae

In this work, we assume geometric deforming of the patches (referred to as shape change in section 3.3 and denoted as S) is only rigid body transformation. Namely we are dealing with 6 DOF (3 for rotation and 3 for translation). In future work, we are aiming at including scale variations and affine transformation into our optimization too.

Rotation is presented by Rodriguez formula. That is rotation in 3D is presented by a vector which has unit length and the direction of vector defines rotations axis (right hand rule). Rotation angle is θ.

$$R(w,\theta) = I_3 + \hat{w}sin(\theta) + \hat{w}^2(1 - cos(\theta)),$$
$$\hat{w} = \begin{bmatrix} 0 & -w_z & w_y \\ w_z & 0 & -w_x \\ -w_y & w_x & 0 \end{bmatrix} \tag{8}$$

Where rotation vector is unit length $[w_x, w_y, w_z]$.

For small θ and by omitting higher order term of $\hat{w}$ we have:

$$R(w,\theta) \approx I_3 + \hat{w}\theta \tag{9}$$

For small changes in rotation, increments are integrated like so;

$$P' = R(w)P_i + T_i \tag{10}$$

Where T is translation in 3D and $P_i = [X_i, Y_i, Z_i]^T$.

By linearizing the system for each of the 6 DOF, we can write the rigid body transformation in a linear system. So above can be re-written as:

$$P_i' = \begin{bmatrix} 0 & Z_i & -Y_i & 1 & 0 & 0 \\ -Z_i & 0 & X_i & 0 & 1 & 0 \\ Y_i & -X_i & 0 & 0 & 0 & 1 \end{bmatrix} q, \tag{11}$$

Where $q = [w_x, w_y, w_z, t_x, t_y, t_z]^T$ is the 6-by-1 vector containing 3 rotation parameters and 3 translation parameters.

Obviously the above is done for a single point in 3D, but we have a regular grid to work with. So naturally the same structure extends.

$$S = \begin{bmatrix} 0 & S_z^1 & -S_y^1 & 1\,0\,0 \\ \vdots & \vdots & \vdots & \vdots\,\vdots\,\vdots \\ 0 & S_z^{n^3} & -S_y^{n^3} & 1\,0\,0 \\ -S_z^1 & 0 & S_x^1 & 0\,1\,0 \\ \vdots & \vdots & \vdots & \vdots\,\vdots\,\vdots \\ -S_z^{n^3} & 0 & S_x^{n^3} & 0\,1\,0 \\ S_y^1 & -S_x^1 & 0 & 0\,0\,1 \\ \vdots & \vdots & \vdots & \vdots\,\vdots\,\vdots \\ S_y^{n^3} & -S_x^{n^3} & 0 & 0\,0\,1 \end{bmatrix} \tag{12}$$

This is S mentioned in 3.

3.5 Global Consistency

So far we have discussed how the new readings from sensor will be broken down to smaller pieces, same size as our patches, and then a reconstructed version of these patches will replace them in the TSDF structure. This means all operations are done locally. But one thing that has not been taken into consideration is that there is correlation between neighboring patches. Therefore we need some sort of mechanism to enforce global consistency.

We take a simple yet effective approach here, and that is while breaking down the TSDF structure into small patches, we allow a significant overlap between neighboring patches. So if a patch has already been replaced by its reconstruction, when we are reading its neighboring patch a part of our reading is coming from what we have already processed (TSDF is update in real-time). So when this new patch is being reconstructed, it naturally tends to adjust itself to reconstruction done for its neighbors. This way global consistency is achieved. In future work we plan to look into more sophisticated ways of imposing global consistency, namely we will make a graph of all the patches and apply some graph optimization techniques.

4 Experiment

4.1 Preparing Training Data with Alignment

Kinect Fusion provides us with a 3D TSDF structure that can be easily broken down to smaller pieces for learning the primitive shapes. In the training set, we take a number of TSDF structures from indoor (office) environment. Each TSDF is produced by running Kinect Fusion for 500 frames. The area reconstructed in each TSDF structure is constrained to a cube of 3-by-3-by-3 m^3 and resolution is 512 voxels in each direction. We found this setting suitable both in terms of level of details reconstructed in final model, and also in terms of memory and

computational requirements. Each voxel has a volume of 6-by-6-by-6 mm^3 and this is a theoretical limit on the resolution of the final model. In practice however, we saw that system is capable of capturing details a lot smaller than that size, up to 3mm in width.

4.2 Learning Phase

Once we have prepared all the sample patches, it is time to learn the more frequent shapes among them. We use Principal Component Analysis for this as explained earlier, 3.3. In our experiments we use patches of 6-by-6-by-6 voxels. Around 400,000 patches are used for training. We keep only a small subset of eigen-shapes, 10 in our case. We saw that on the train-set, keeping only 10 eigen shapes introduces an error of less than 5%. Table 1 shows the top 20 eigen-value, one can see that eigen values drop very rapidly, indicating the presence of structure in patch appearance.

Attention needs to be paid to choosing the right number of eigen-shapes (size of matrix A in 2). By using higher number of eigen-shapes, we give our model more flexibility. This flexibility will allow the model to match the (noisy) data better but that does not necessarily translate to better approximation of noise-free structure. On the other hand, by giving too few eigen-shapes to the model, it will fail to reconstruct the more difficult structure that may be present in the scene. Choice of number of eigen-shapes to keep depends very much on size of patches, sensor noise and other problem specific factors. Our choice was made based on experimentation.

Table 1. Top 20 eigen values of training set

Number	1	2	3	4	5	6	7	8	9	10
Eigen Value	19.21	8.93	6.10	5.08	3.56	2.73	2.33	1.96	1.89	1.58
Number	11	12	13	14	15	16	17	18	19	20
Eigen Value	1.42	1.21	1.17	1.07	1.01	0.92	0.81	0.76	0.73	0.67

4.3 Fitting

In this phase we use Kinect Fusion again for obtaining test data. But only the first frame reading from the sensor is considered and Kinect Fusion stops, after receiving this first frame. We used fixed number of 10 iteration in the iterative optimization phase, However we also observed that because of the pre-alignment there is very little transformation correction needed and Convergence is very quick. Algorithm 1 outlines the systems flow.

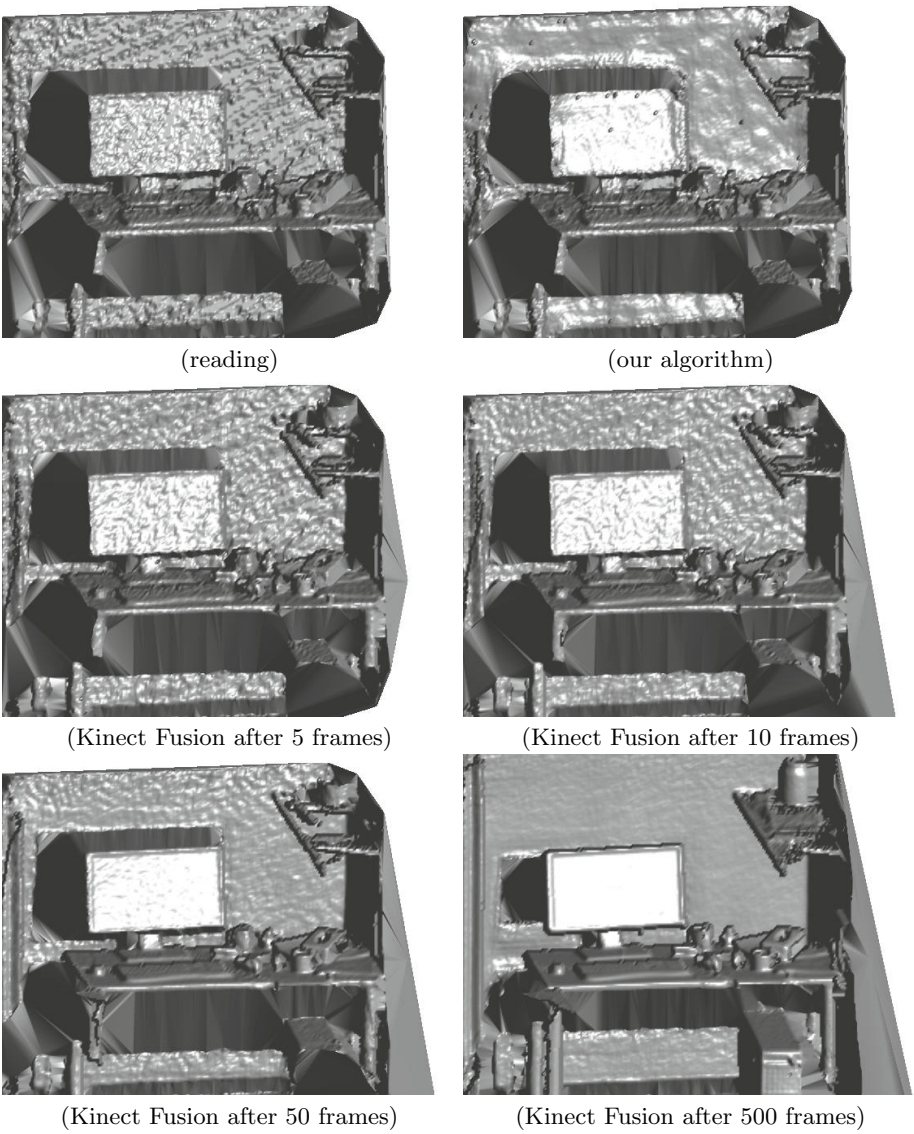


Fig. 5. Top left is the original reading, top right is the output of our algorithm. Row 2 and 3; shows the output of kinect fusion after fusing data from 5,10, 50 and 500 frames

```
initialization;
if training is needed then
    read training data;
    pre-processing;
    form a big matrix and apply PCA;
    trim the eigen-shapes (A) and calculate the jacobian for corresponding A;
end
load eigen-shapes and jacobians;
read new test data;
break TSDF down into small patches;
pre-process;
for each patch in test set do
    project the reading into space span by A;
    for 10 iterations do
        correct for rigid-body transformation;
        project into space span by A;
        update parameters p and c;
    end
    integrate the reconstructed data back into original TSDF;
end
```

Algorithm 1: Pseudocode of our algorithm

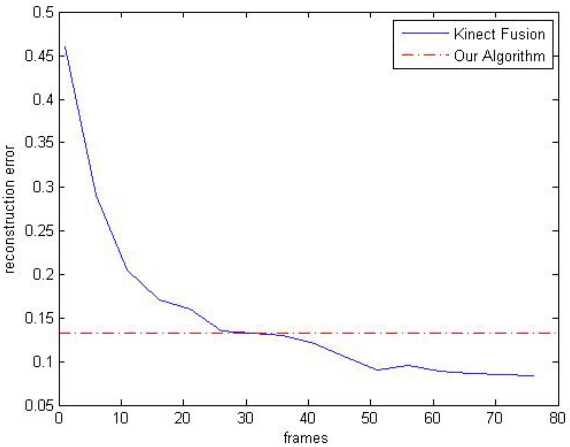


Fig. 6. Reduction of reconstruction error due to merging more frames together in Kinect Fusion (blue), or applying our method to one frame only (red).

4.4 Evaluation

Figure 5 shows the results of a test TSDF, and the reconstructed map. One can see visually that while 3D map has been denoised, most of 3D structure has been

preserved. Output of our algorithm is compared with output from Kinect Fusion after 5, 10 and 50 frames.

Also figure 6 demonstrates the reduction in reconstruction error when different number of frames have been used in Kinect Fusion, (steps of 5 frames). Measure of error used here is the root-mean-square between difference of ground truth and reconstructed patch.

In our experiment denoising achieved by 'projection alone' was equal to fusing data from around 8 frames. This might be useful if one has to sacrifice the optimization step in the interest of time or computational power available. But by adding 10 iteration in 'optimization step' reduction in error is equivalent to 28 frames combined using Kinect Fusion. Figure 6.

5 Conclusion

In this work we introduced a generative model for dense 3D maps, based on deformable 3D patches. Our model is capable of learning prominent spatial structure in man-made environments in an unsupervised manner. We also introduced a fitting strategy which jointly optimizes for 3D appearance and geometric transformation using robust Gauss-Newton method. We showed that significant noise reduction and compactness can be achieved using our method. We plan to integrate our method directly inside a SLAM loop and also employ better global optimization methods in our future work.

References

1. Curless, B., Levoy., M.: A volumetric method for building complex models from range images. In: Proceedings of the 23rd Annual Conference on Computer Graphics and Interactive Techniques (ACM) (1996)
2. Davison, A.J., Reid, I.D., Molton, N.D., Stasse, O.: Monoslam: Real-time single camera slam. IEEE Transactions on Pattern Analysis and Machine Intelligence **29**(6), 1052–1067
3. Hejrati, M., Ramanan, D.: Analysis by synthesis: 3d object reconstruction by object reconstruction. In: Computer Vision and Pattern Recognition (CVPR) (2014)
4. Klein, G., Murray, D.: Parallel tracking and mapping for small ar workspaces. In: 6th IEEE and ACM International Symposium on Mixed and Augmented Reality, ISMAR (2007)
5. Montemerlo, M., Thrun, S., Koller, D., Wegbreit, B.: FastSLAM: A factored solution to the simultaneous localization and mapping problem. In: Proceedings of the AAAI National Conference on Artificial Intelligence, AAAI, Edmonton, Canada (2002)
6. Newcombe, R.A., Izadi, S., Hilliges, O., Molyneaux, D., Kim, D., Davison, A.J., Kohi, P., Shotton, J., Hodges, S., Fitzgibbon, A.: Kinectfusion: real-time dense surface mapping and tracking. In: 2011 10th IEEE International Symposium on Mixed and Augmented Reality (ISMAR), pp. 127–136, October 2011
7. Newcombe, R.A., Lovegrove, S., Davison, A.: Dtam: dense tracking and mapping in real-time. In: 2011 IEEE International Conference on Computer Vision (ICCV), pp. 2320–2327, November 2011

8. Salas-Moreno, R., Newcombe, R., Strasdat, H., Kelly, P., Davison, A.: Slam++: Simultaneous localisation and mapping at the level of objects. In: 2013 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), pp. 1352–1359, June 2013
9. Tzimiropoulos, G., Pantic, M.: Optimization problems for fast aam fitting in-the-wild. In: 2013 IEEE International Conference on Computer Vision (ICCV), pp. 593–600. IEEE (2013)
10. Tzimiropoulos, G., Pantic, M.: Gauss-newton deformable part models for face alignment in-the-wild. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, pp. 1851–1858 (2014)
11. Zhang, Z.: Iterative point matching for registration of free-form curves and surfaces. International Journal of Computer Vision **13**(2), 119–152 (1994)
12. Zhu, Y., Zhang, Y., Yuille, A.L.: Single image super-resolution using deformable patches. In: IEEE Conference on Computer Vision and Pattern Recognition (CVPR) (2014)