SLAM Homework 5 - TSDF

Runze Yuan (yuanrz@shanghaitech.edu.cn)

November 28, 2021

1. Algorithm Pipeline

TSDF fusion is a fusion method to combine the different views of the observation to construct a smooth map

The algorithm is as below

1. Construct a map of size 0.2x0.2x0.2(m), the voxel size we set is 0.01m. Then there will be 8000 voxels in the map.
2. Given each depth image with a pose, first we represent every voxel as the center point of the voxel. Transform the point into the camera frame using the pose. Then project it into the image.
3. If the projection of the voxel is inside the image, we try to update the tsdf value in the voxel. First, we need to calculate the scale, since the tsdf value is the depth projected into the principle axis.

$$\lambda = ||K^{-1} \begin{bmatrix} u \\ v \\ 1 \end{bmatrix} ||_2$$

4. Then we should calculate the sdf value, the sdf value should be positive if the voxel is in front of the surface, vice versa. Indicate the voxel point in the camera frame as P_c, the projection into the image as u, and the depth image as $R()$So the sdf value should be

$$sdf = ||P_c||_2/\lambda - R(u)/scale$$

   where the scale is the depth scale constant

5. The sdf value should be truncated, here we set the truncated margin(μ) as voxel_size. Then the voxel which is too far or too close won't be considered.

$$tsdf = \begin{cases} \min(1, \frac{sdf}{\mu}) sign(sdf), & sdf > -\mu \\ null, & else. \end{cases}$$

6. Last, we update tsdf value and the weight in that voxel. Here we set the weight of every voxel is 1.

$$tsdf = \frac{w_{last} * tsdf_{old} + tsdf_{new}}{w_{last} + 1}$$
$$w = w_{last} + 1$$

2. Usage

1. Dependency:

   (a) python 3.7
   (b) opencv-python (for reading image)
   (c) numpy (for calculation)
   (d) open3d (for visualization)
   (e) skimage (for matching cube)

2. run: **python TSDF.py**

Here we provide a class named TSDFMap which contains

1. Attributes:
   (a) map: a 0.2x0.2x0.2 voxel space containing tsdf value
   (b) weight: a 0.2x0.2x0.2 voxel space containing weight value
2. method:
   (a) update_map: given depth image and pose, this method is used to update our tsdf map
   (b) visualize: using marching cube to visualize the mesh

3. Evaluation

Time efficiency:

The time cost is shown as below

Table 1: performance table

	time cost(s)	time per frame(s)
tsdf	45.4921	3.2494

We can see that the time cost of our algorithm is quite high. The reason might be that we calculate the projection of all the voxels, which is quite time costing. What's more, the data structure we used is not follow the alignment principle. An alternative way is that we compute the FOV of the camera, and only iterative the voxels in FOV, which may be more complex but more efficient.

Also, using the hash table to represent the voxel space is a better method to accelerate.

Reconstruction:

The reconstruction result is shown as below, on the left is the point cloud model we reconstruct using the depth image, and one the right is the mesh we construct using tsdf.

We can see that the mesh is not much smooth. The reason might be that the voxel size is too large. Also, we set the weight of every tsdf value as 1 naively, which may be another reason of the bad reconstruction.

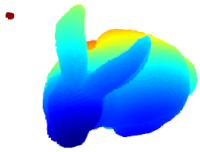

Figure 1: ground truth (top)

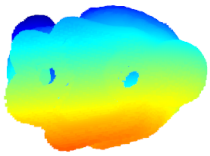

Figure 2: ground truth (rear)

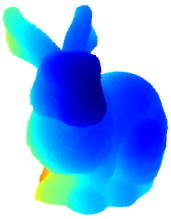

Figure 3: ground truth (front)

Figure 4: ground truth (back)

Figure 5: ground truth (left)

Figure 6: ground truth (right)

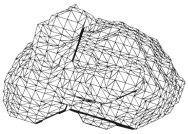

Figure 7: construction (top)

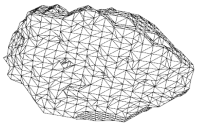

Figure 8: construction (rear)

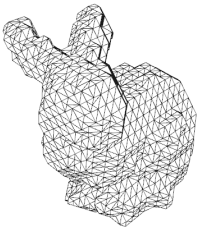

Figure 9: construction (front)

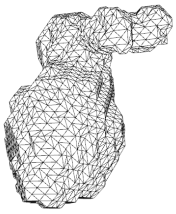

Figure 10: construction (back)

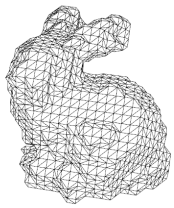

Figure 11: construction (left)

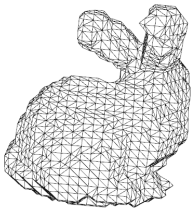

Figure 12: ground truth (right)