Simultaneous Localization and Mapping using Monte Carlo Localization for 3D Scene creation

Nitin J. Sanket School of Engineering and
Applied Science
University of Pennsylvania
Email: nitinsan@seas.upenn.edu

***Abstract*—This project presents an approach for mapping a scene while localizing the robot (SLAM) using a 2D laser scan. The computed 2D pose is then used to generate a 3D point cloud map of the world using the RGBD data on the onboard kinect sensor.**

I. Problem Statement

The aim of the project was to perform simultaneous localization and 3D mapping using 2D laser scans.

II. Data Pre-processing

A. Ground Hits Removal

When the robot's head is tilted with respect to the ground, some laser points hit the ground and they over/under-estimate the distance and they need to be compensated. In effect the plane of the scan is sliced by the ground and the hits on the line which is the intersection of the scan plane and ground plane need to be removed. The equation used for this process is given below:

$$Scan \sin(\theta)\cos(\gamma) \geq H_{head} + (H_{Lidar} - H_{head})\cos(\theta)$$

here, θ represents the head pitch angle and γ represents the scan angle, scan is the actual distance.
The correction for the skew in the laser scan plane is also accounted for so that the scan plane can be made parallel to the ground plane by the following formula

$$Scan = Scan\cos(\theta)$$

here again θ is the head pitch angle. Refer to Fig. 1 for a plot showing the ground hit removal. Also, the hits which are smaller than 0.5m and larger than 30m are removed as they are junk values/values outside the sensor range.

B. Depth Image Filtering

The depth camera on the kinect v2 has severe distortion around the corners. So, using the the distortion parameters from the calibration procedure to undistort the image is a logical step. However, because the way the depth camera is constructed normal undistortion routines do not work and they give junk values. A simple illustration of this is given in Fig. 2.

To remove some of the junk values due to serve corner distortion I experimented with two different kinds of blur: Median Filter and Anisotropic Diffusion. Median Filter just picks the median value in a window. This is generally recommended for depth images because the depth values sparsely jump around but the median around a patch is generally more accurate. However, the median filter sometimes blurs out thin edges. To avoid this, I used Anisotropic Diffusion which did not work well because the noise removal by this filter was minimal. So the simplest filter (median filter) performed the best. A simple illustration of this is given in Fig. 3.

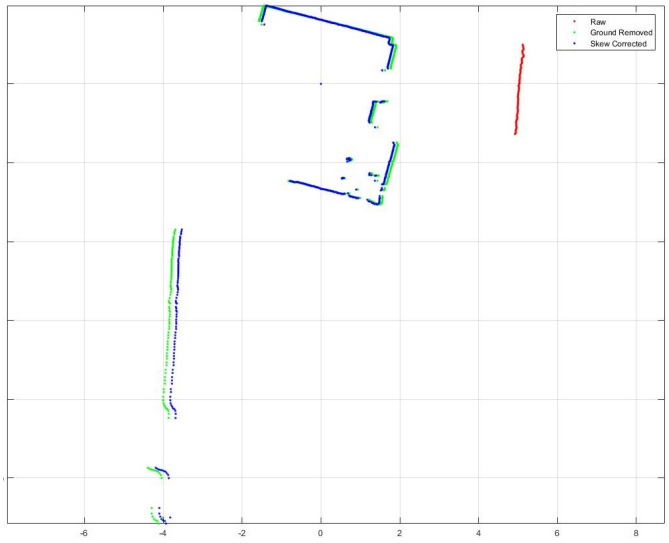

Fig. 1: Scan Correction pipeline outputs. (most of the red hits are behind green and blue ones as they overlap)

III. Simultaneous Localization and Mapping (SLAM)

A. Monte-Carlo Localization (MCL)

The overall algorithm for MCL [1] is described in Algorithm 1.

B. Occupancy Grid Mapping (OGM)

The overall algorithm for OGM [2] is described in Algorithm 2. The overall SLAM algorithm is described in 3 and is a combination of MCL and OGM.

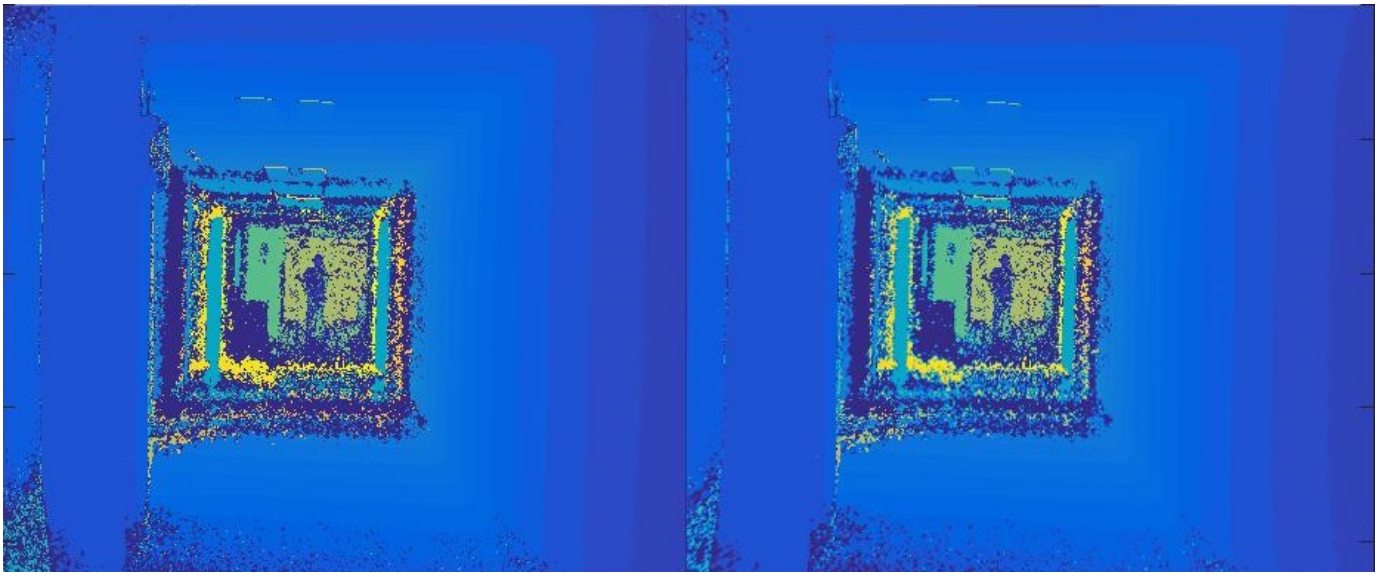

Fig. 2: Left: Raw Depth Image, Right: Undistorted Depth Image.

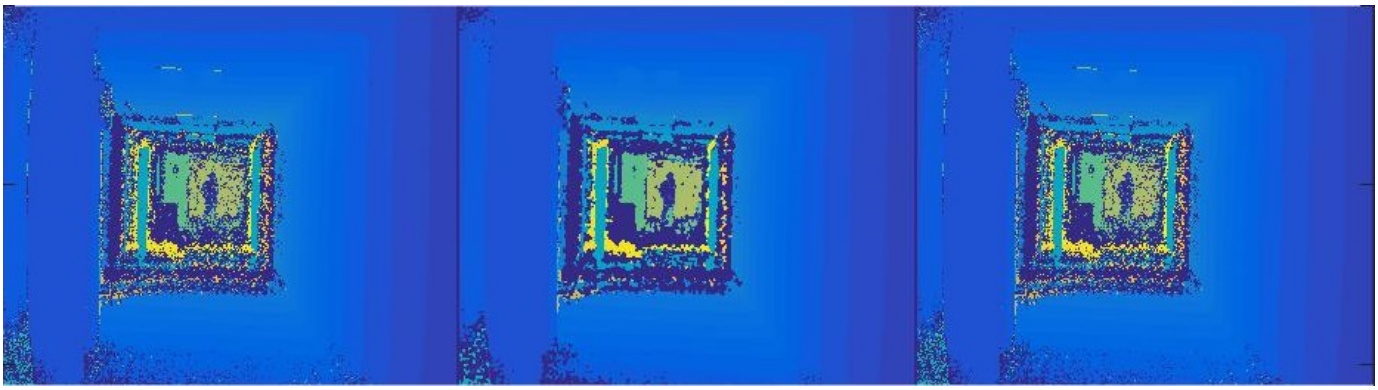

Fig. 3: Left: Raw Depth Image, Middle: Median Filter Output, and, Right: Anisotropic Diffusion Output.

IV. 3D-Map Creation with Ground Detection

A. 3D-Map

First, compute the 3D co-ordinate in depth camera frame using,

$$x^{IR} = \frac{uz}{f_x^{IR}}$$

$$y^{IR} = \frac{vz}{f_y^{IR}}$$

Here, u and v are centered pixel co-ordinates. z is the depth value for the current pixel.
Next, transform these points into RGB camera frame using

$$X^{RGB} = RX^{IR} + T$$

where $X^{IR} = \begin{bmatrix} x^{IR} & y^{IR} & z^{IR} \end{bmatrix}$ Next, re-project the points onto the image plane

$$x^{RGB} = \frac{uz}{f_x^{RGB}}$$

$$y^{RGB} = \frac{vz}{f_y^{RGB}}$$

Lastly, read the color (r, g, b) at $(u, v)^{RGB}$.

To obtain the 3D-Point cloud in global frame the following equations are used,

$$X^{RGB}{}_W = \begin{bmatrix} 0 & 0 & 1 \\ -1 & 0 & 0 \\ 0 & -1 & 0 \end{bmatrix} X^{RGB}$$

This is needed to change the co-ordinates from camera frame to robot body frame.
The underlying equations give the 3D point cloud in world frame.

$$X^{RGB}{}_W = RX^{RGB} + \begin{bmatrix} PoseX \\ PoseY \\ PoseZ \end{bmatrix}$$

$$R = \begin{bmatrix} \cos\psi & -\sin\psi & 0 \\ \sin\psi & \cos\psi & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \cos\theta & 0 & \sin\theta \\ 0 & 1 & 0 \\ -\sin\theta & 0 & \cos\theta \end{bmatrix}$$

B. Ground Detection

Let us say the ground normal is denoted by η. If we define the camera axis is y down, z through camera axis and x left. In the camera frame, when the camera has some pitch we see

Data: NumParticles, Odom(t), Obs(t), GMAP, NoiseParams, NEffThld
Result: BestPose
Initialize All ParticlePoses to zeros **for** $\forall t$ **do**
  % Compute incremental change in state
  ΔState = Odom(t)-Odom(t-1)
  for $\forall particlesp$ **do**
    % Motion Model Update
    State(p,t+1) = State(p,t) + ΔState + Noise
    % Transform Local Frame to Global Frame
    StateGlobalFrame(p,t+1) = f(State(p,t+1))
    % f is the transformation function to transform from local to global frame
    % Compute correlation between GMAP and LMAP
    c(p) = CompCorr(LMAP, GMAP)
    % Re-weigh the particles
    w(p,t+1) = w(p,t)*c(p)
  end
  % Re-normalize weights
  $w(p) = \frac{w(p)}{\Sigma(w)}$, $\forall p$
  if $N_{eff} = \frac{\left(\sum_p {w_p}^2\right)^2}{\sum_p (w_p)^2} <= NEffThld)$ **then**
    % Re-sample Particles
    State(p,t+1) = ResampleParticle(State,w)
    % Make all weights equal
    w(p,t+1) = 1/NumParticles, $\forall p$
  end
  % Find Best Particle as state estimate using max weight
  EstimatedState(t+1) = State corresponding to max(w(:,t+1))
end

Algorithm 1: Monte Carlo Localization

the ground normal being transformed according to the equation below:

$$\eta_{ground} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos\theta & -\sin\theta \\ 0 & \sin\theta & \cos\theta \end{bmatrix} \begin{bmatrix} 0 \\ -1 \\ 0 \end{bmatrix}$$

These serve as the coefficients a_0, a_1 and a_2. The fourth coefficient is $a_3 = h$ (height of the robot).

$$h = 1260 + 70\cos\theta$$

Now, the ground plane model is given by,

$$a_0x + a_1y + a_2z + a_3 = 0$$

Here,

$$x = \frac{uz}{f_x}$$

$$y = \frac{vz}{f_y}$$

Here, u and v are centered pixel co-ordinates. z is the depth value for the current pixel.

Data: LogOddOcc, LogOddFree, Pose, GMAP, LMAP, SatLow, SatHigh
Result: Updated GMAP
% Transform LMAP to GMAP using Pose, f is the transformation function
TransformedLMAP = f(LMAP, Pose)
% Occupied Cells and Free Cells are found from TranformedLMAP
% Update Occupied Cells
GMAP(OccupiedCells) += LogOddOcc
% Update Free Cells
GMAP(FreeCells) += LogOddFree
% Saturate Values
if *GMAP(i,j)¿SatHigh*
then
  GMAP(i,j) = SatHigh
end
if *GMAP(i,j)¡SatLow*
then
  GMAP(i,j) = SatLow
end

Algorithm 2: Occupancy Grid Mapping

Data: RawData
Result: RobotPose,Updated GMAP
Initialize required parameters
for $\forall t$ **do**
  MCL
  OGM
end

Algorithm 3: SLAM using MCL and OGM

Now, a valid ground point should satisfy,

$$|a_0x + a_1y + a_2z + a_3| < \varepsilon$$

As, I am not accounting for body pitch using a rotation matrix, I account for the small pitch in the body by saying all the points below a certain threshold (10cm in my case) should be ground points. This also includes steps sometimes. The ground point validity check is therefore given by,

$$a_0x + a_1ya + a_2z + a_3 \leq \varepsilon'$$

These co-ordinates are converted to grid units and RGB pixel value is copied.

V. Correlation

A. Using Cross-correlation for map correlation

Matlab's built-in corr2 computes cross-correlation between 2 real matrices/images, so this can be used to compute correlation in logOdds. This seemed to work well and was robust to noisy measurements but was too slow (5 times slower than the C++ code given).

B. AND based correlation

Both the local and global maps were converted to binary and a simple AND operation was performed to find the number of common occupied pixels. However, this is highly susceptible to pixel shifts, the wall starts drifting due to this problem.

C. Line Segment Detector based correlation

A line segment detector [3] was used to detect the line segments in the current scan and the thresholded global map (binary after thresholding). A sample plot of the current map, current scan (red highlight) and detected line segments are shown in Fig. 4. The correlation metric was defined as follows:

$$c = \sum_{j} \min \sum_{i,j \neq i} a_i \times b_j + ed(a, b)$$

Here a and b are line segments represented as vectors, ed represents the euclidian distance between centroids of the line segments. The idea is take a line segment and compute cross products with all other line segments, find the minimum and add it to the list. Now exclude these pairs of lines and repeat. Final sum is the correlation value. The idea is to sort of hack a ICP algorithm without explicitly doing it. The code was very slow and had some wierd artifacts maybe because I was combining cross-products and norms as a sum. Fig. 4 was the output for train dataset 1.

VI. Effect of parameters on Particle Filter

A. Number of Particles

Decreasing the number of particles affects the result especially for longer datasets. With very few particles, the grid area explored per timestep is small hence a highly sub-optimal minima is found even if there is minimal amount of drift. This basically reflects how badly sampling a distribution can not even resemble the distribution itself. To overcome this, a variant of re-sampling called low-variance sampling can be used which is discussed in the next section. Also, increasing number of particles increases run-time in at-least $\mathcal{O}(n)$.

B. Noise Parameters

The motion model noise tells to what degree the trust the odometry versus the scan matcher. A huge noise on position places more trust on the scan matcher which will hugely underestimate distances in a straight path. A small noise on angle will make the angles to be underestimated/overestimated i.e., the right angle turns will look obtuse/acute. Also, if angle noise is high the scan matcher might take a wrong turn completely. A careful balance of reasonably low position noise and reasonably high angle noise give good results. The parameters used were (0.01m,0.01m,0.05 degrees).

C. Map Resolution

A map resolution of 10cm was used for all experiments for speed an accuracy. A finer resolution gave better results for some datasets but was marginally slower. A coarser resolution seem to make the angles diverge a lot.

D. Sub-sampling Scans

Sub-sampling scans helped avoid jumps in data for updation. This also sped up the execution by a huge factor. Too much sub-sampling makes the code go haywire as the motion model falls apart.

E. Log-Odds values and their corresponding saturation thresholds

The Log-Odds values and saturation thresholds control how fast the cells will be occupied (needed for obstacle avoidance) or cleared (needed when there are moving obstacles). Generally, the lidar is more certain about the wall than the free space hence the update value for occupied was about 4 times higher than that for free cells. This helped get near-perfect loop closure without any explicit loop-closure algorithms.

VII. Re-sampling strategy

Two different methods for re-sampling particles were used.

A. Random Sampling

A random number was picked between 0 to 1 (weights are normalized). All the weights are arranged in such a way that they sum upto 1 and their width on the number line is proportional to their magnitude. Cumulative sum is used to pick a particle index based on the random number. This strategy seems to model the distribution well when there are fairly large number of particles like 50 or 100. This sampling strategy falls apart when there are very few particles. The output tends to follow the odometry data if the noise is low and jumps around unstably when noise is high.

B. Low-Variance Sampling

A random number is picked between 0 to 1 (weights are normalized). All the weights are arranged in such a way that they sum upto 1 and their width on the number line is proportional to their magnitude. In a modulo fashion and uniform increments sample the number line and look up the particles associated with it. This guarantees that the best particle is picked. Hence this method works very well for low number of particles.

Refer to Fig. 5. One can clearly observe the uncertainty in random sampling with high noise (left-most figure) and huge drift (middle figure) and reasonably good results with low-variance sampling (right-most figure).

VIII. Using IMU heading vs Odometry heading

Generally the IMU is more accurate for incremental angular updates than the odometry. This proved to work well for all the training datasets with lower noise. But for the test set it did not have near-perfect loop closure. Odometry heading with a slightly larger noise was almost at par or slightly better for all the sets, hence I used Odometry heading. A sample plot of dead-reckoning using Odometry heading and IMU heading is shown in Fig. 6, they have approximately the same shape but the values are significantly different.

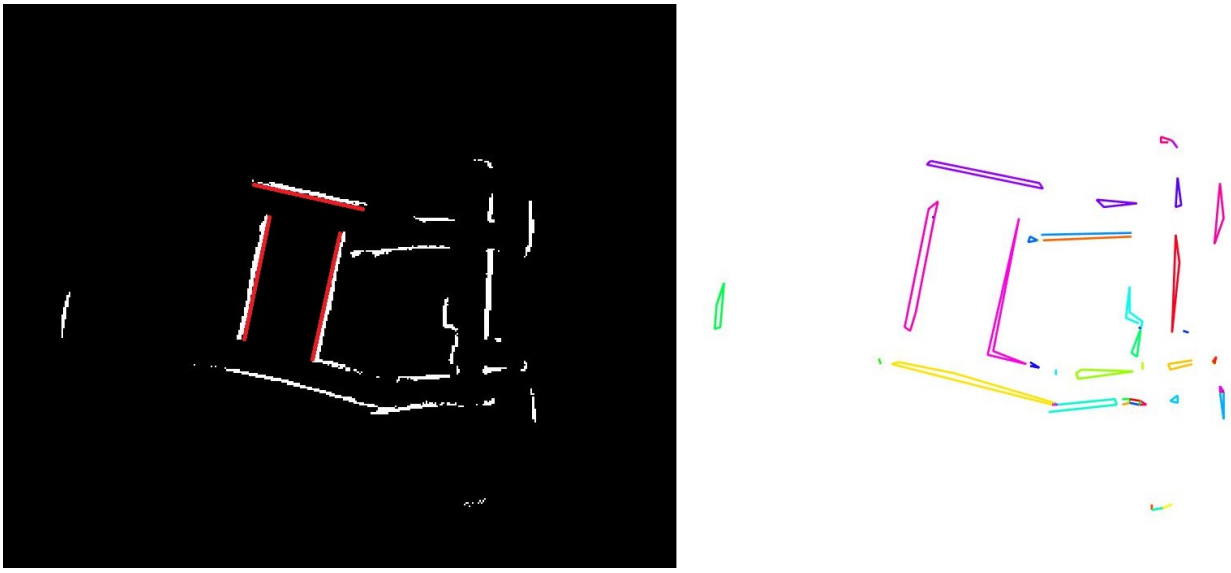

Fig. 4: Line Segment Detection on thresholded global map.

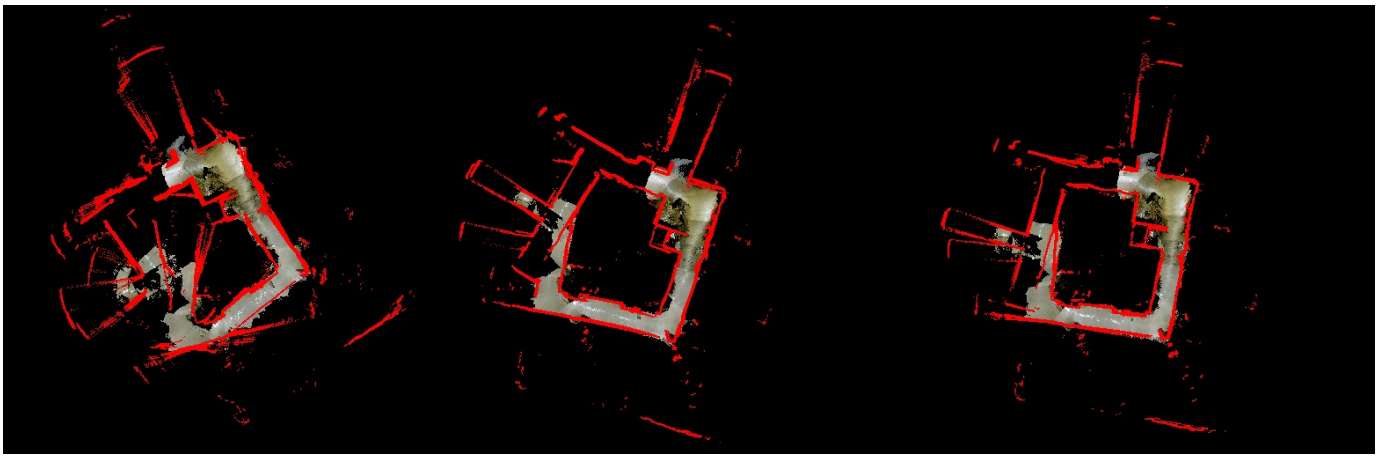

Fig. 5: Left to Right: Random Sampling with high noise, Random Sampling with low noise and Low-Varience Sampling, all with 10 particles.

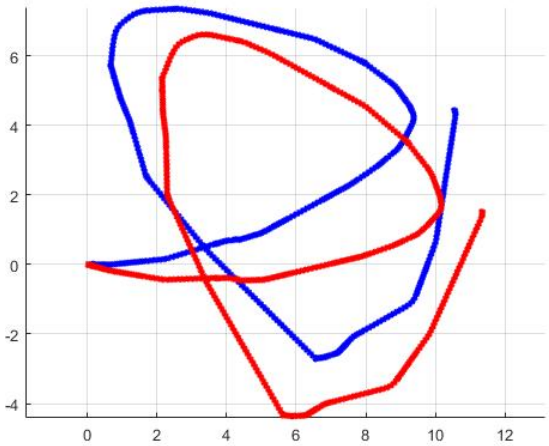

Fig. 6: Dead-reckoning using odometry heading (blue) and RPY (red).

IX. Efficient Data Handling

The point cloud files were huge and hard to handle on a laptop with 12GB of RAM. Hence, the mat files were stripped and frames were saved as images. The point cloud data for each file was saved as a mat file (4 to 6GB per set in total). The point cloud data was subsampled temporally and plotted for easy visualization and rotation purposes. The script to convert the provided mat files to jpg images is provided and is almost fully independent of codecs.

X. Other interesting things I tried

I also tried running ORB-SLAM2, the current state-of-the-art monocular SLAM algorithm on the dataset provided to us. The algorithm failed miserably due to low frame rate and inconsistent frame rate/ frame-drops. I couldn't even begin localizing which demonstrates that the usage of a laser is feasible due to huge range and high frame rate and seemingly lower computation cost during scan matching.

XI. Results

A. Train Set

The occupancy grid for Map 0 is shown in Fig. 7 and the textured floor is shown in Fig. 8, the 3D-map for Map 0 is shown in Fig. 9. The occupancy grid for Maps 1, 2 and 3 are shown in Figs. 10, 11, 12 respectively. The textured floor and 3D-map for Map3 are shown in Figs. 13 and 14 respectively.

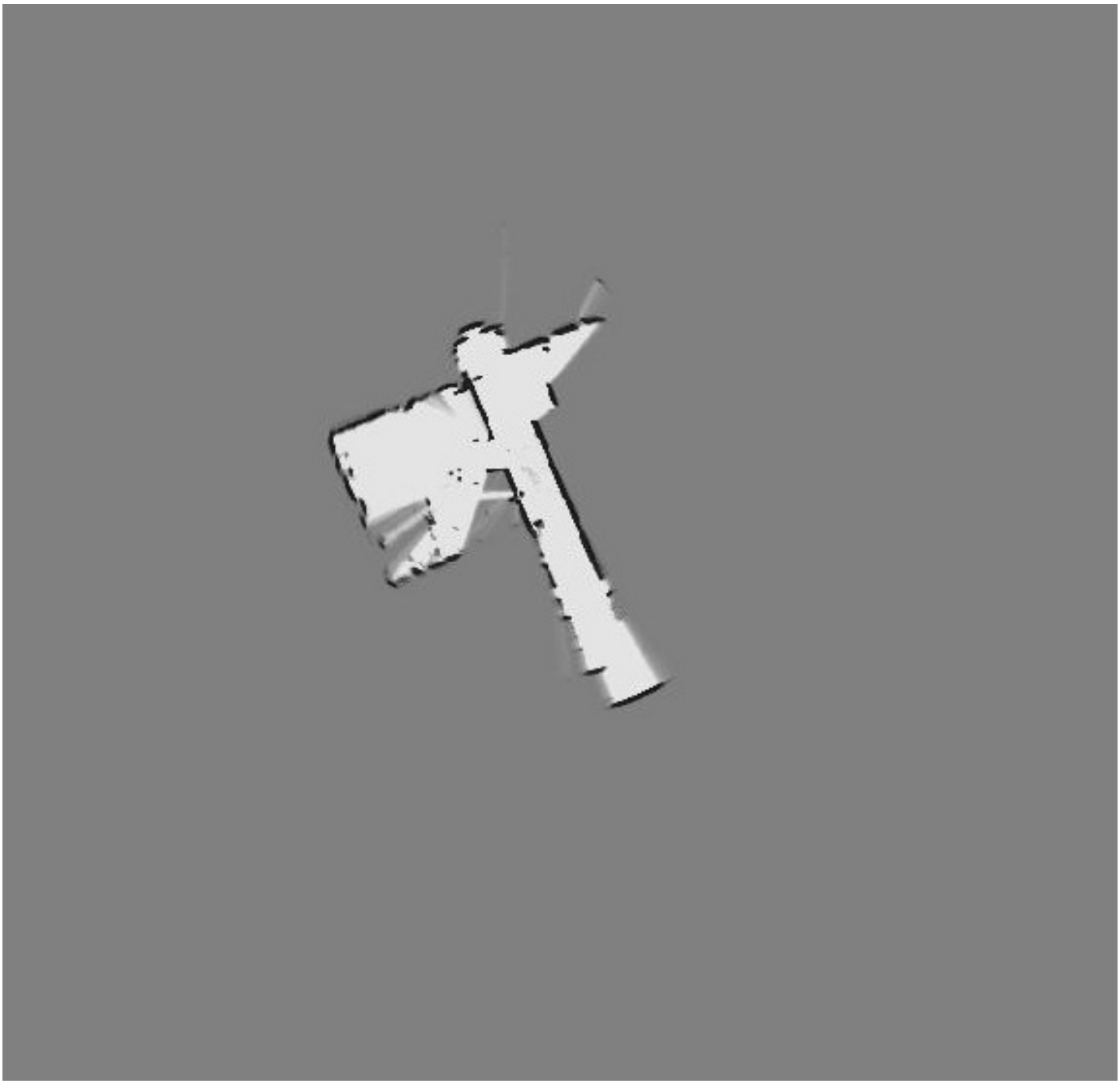

Fig. 7: Occupancy Grid for Map0.

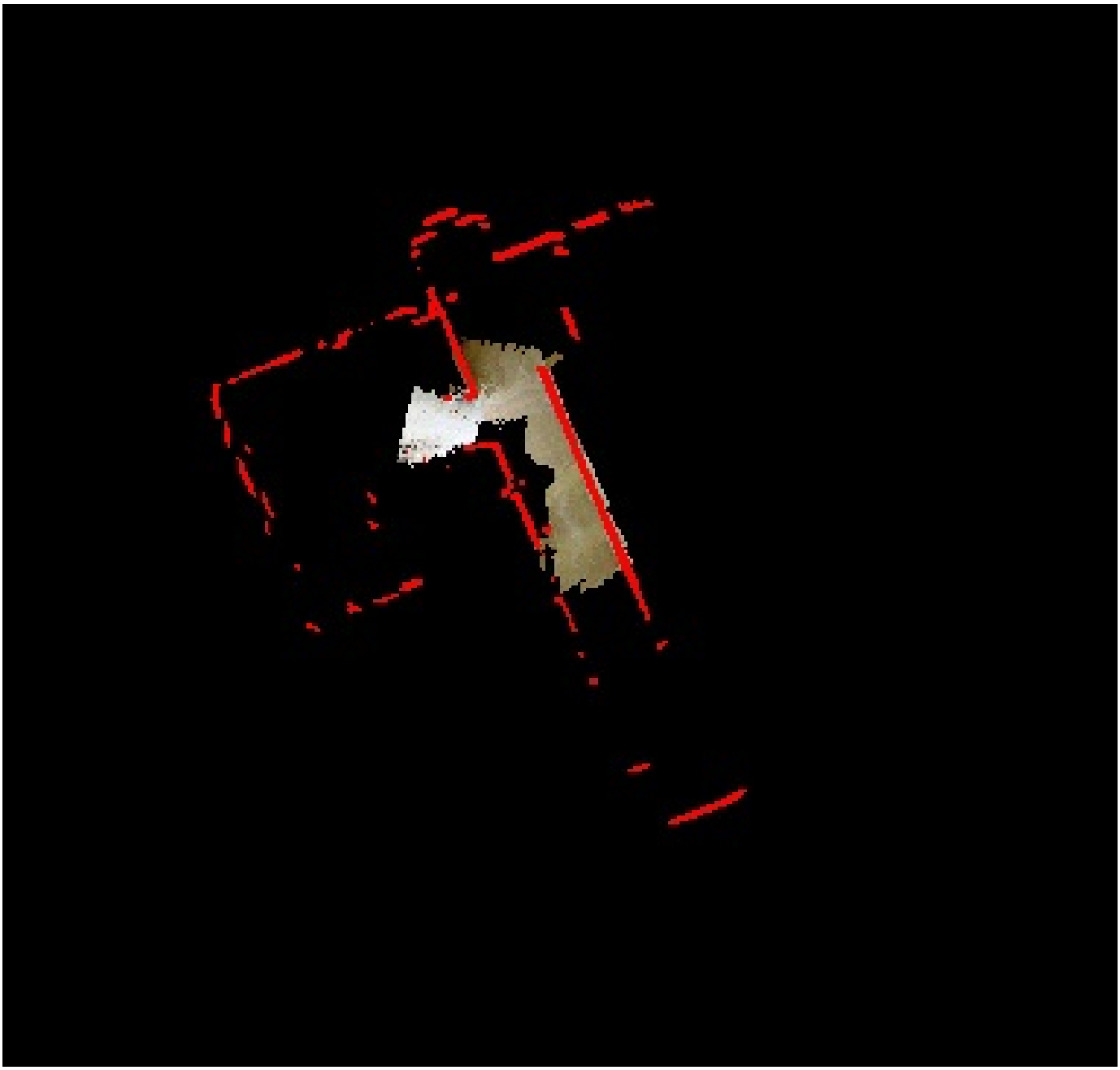

Fig. 8: Textured floor for Map0, red highlights show walls.

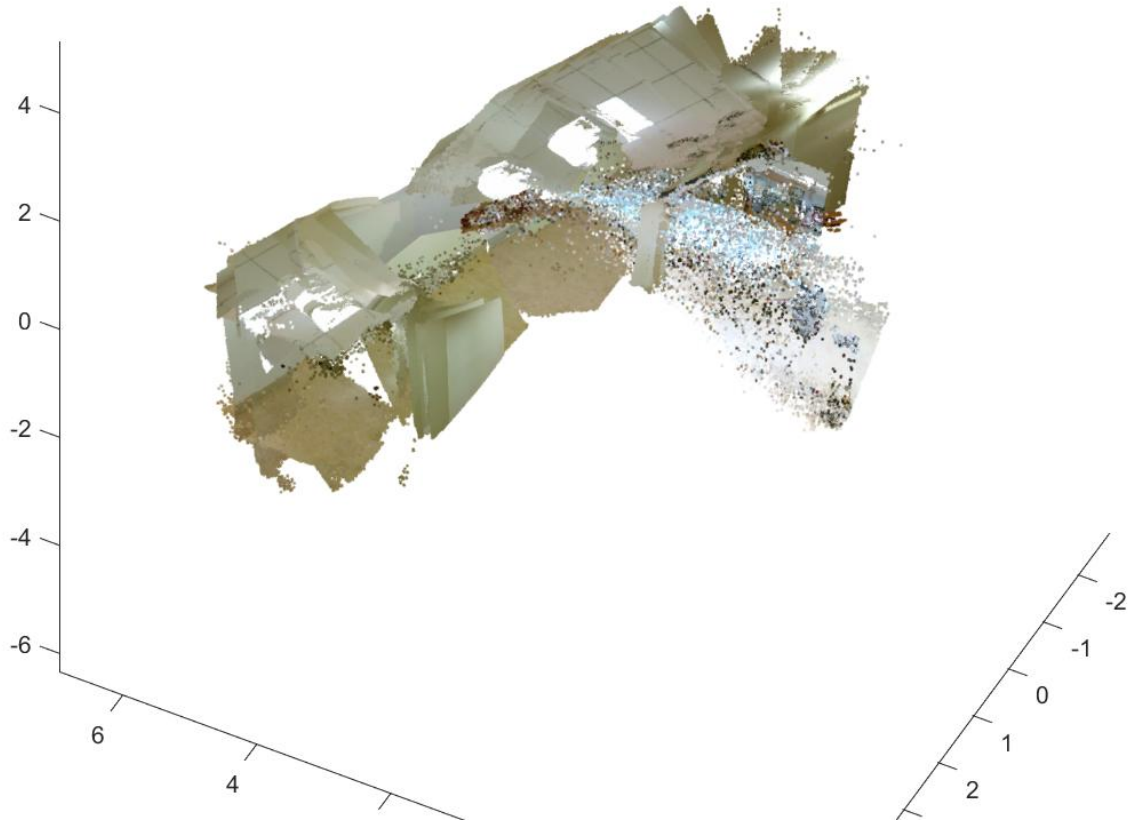

Fig. 9: 3D-map for Map0.

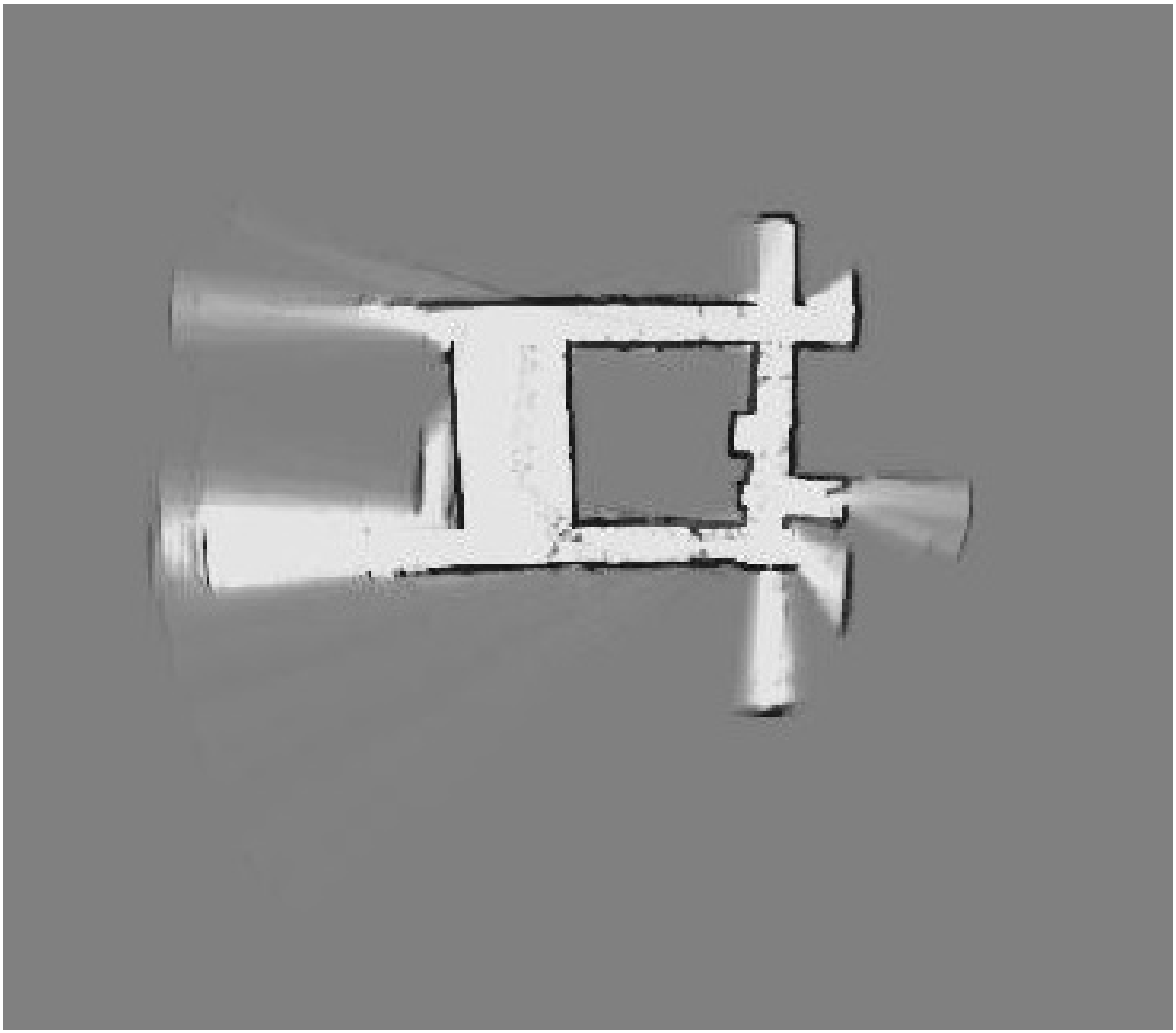

Fig. 10: Occupancy Grid for Map1.

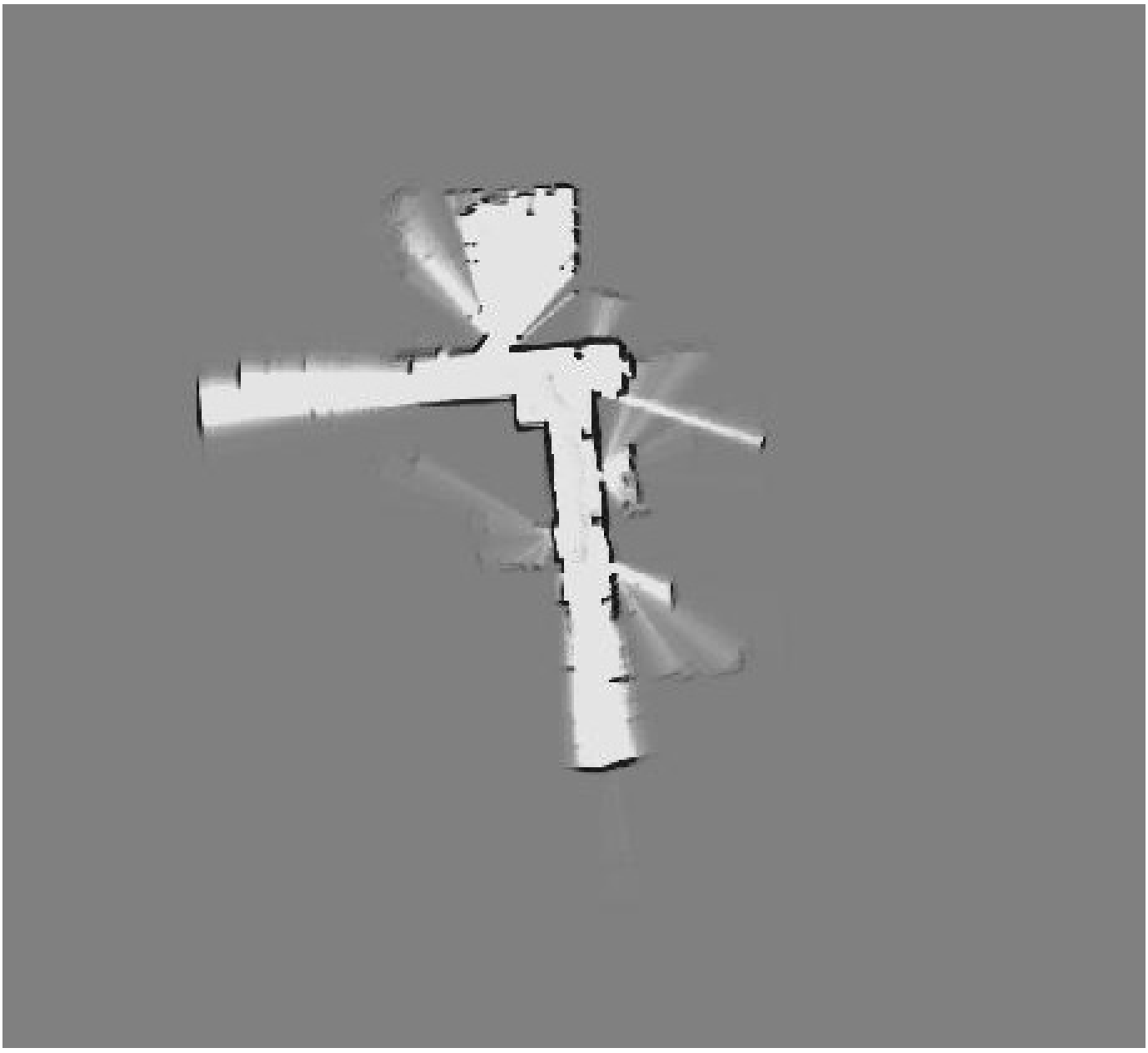

Fig. 11: Occupancy Grid for Map2.

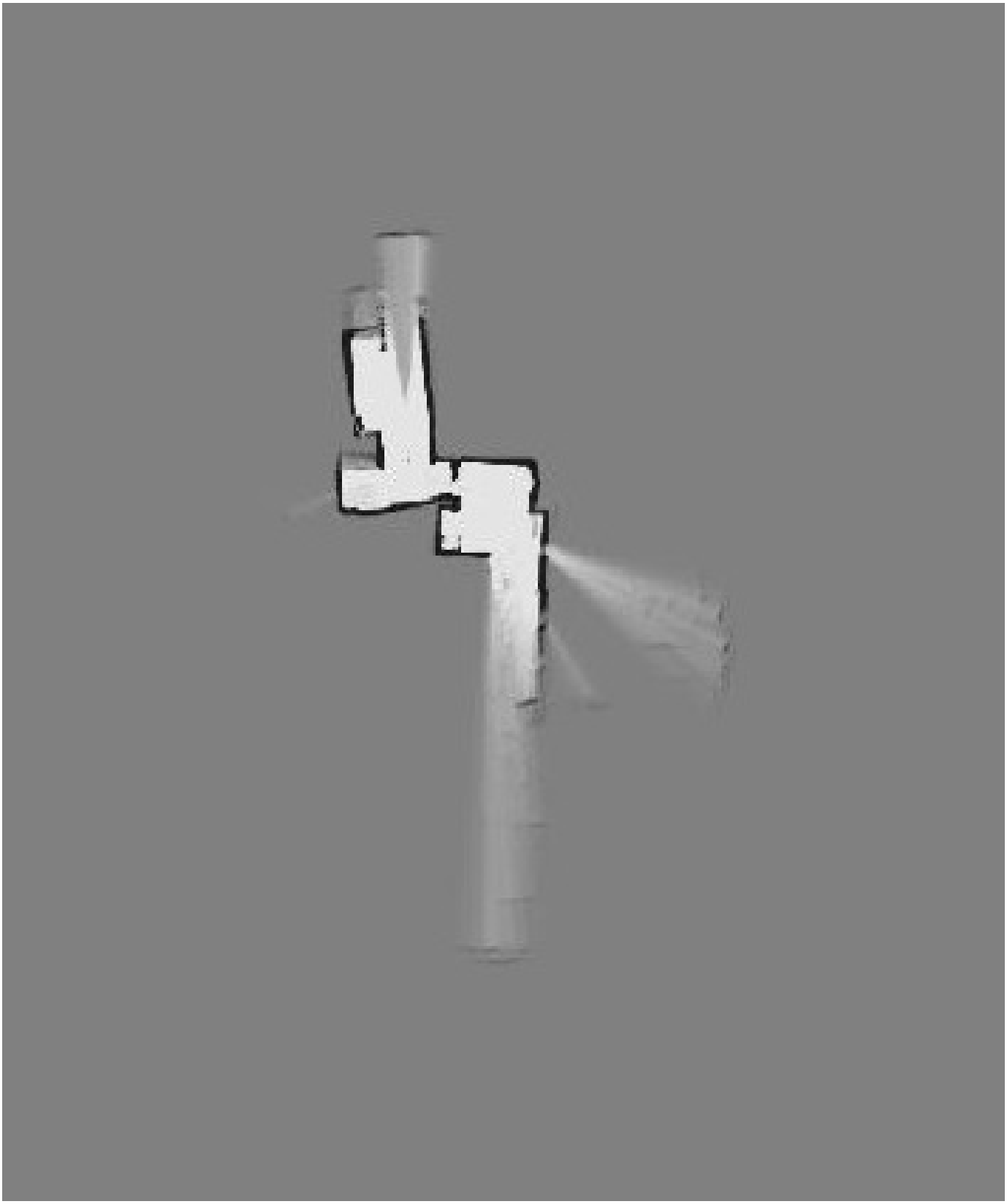

Fig. 12: Occupancy Grid for Map3.

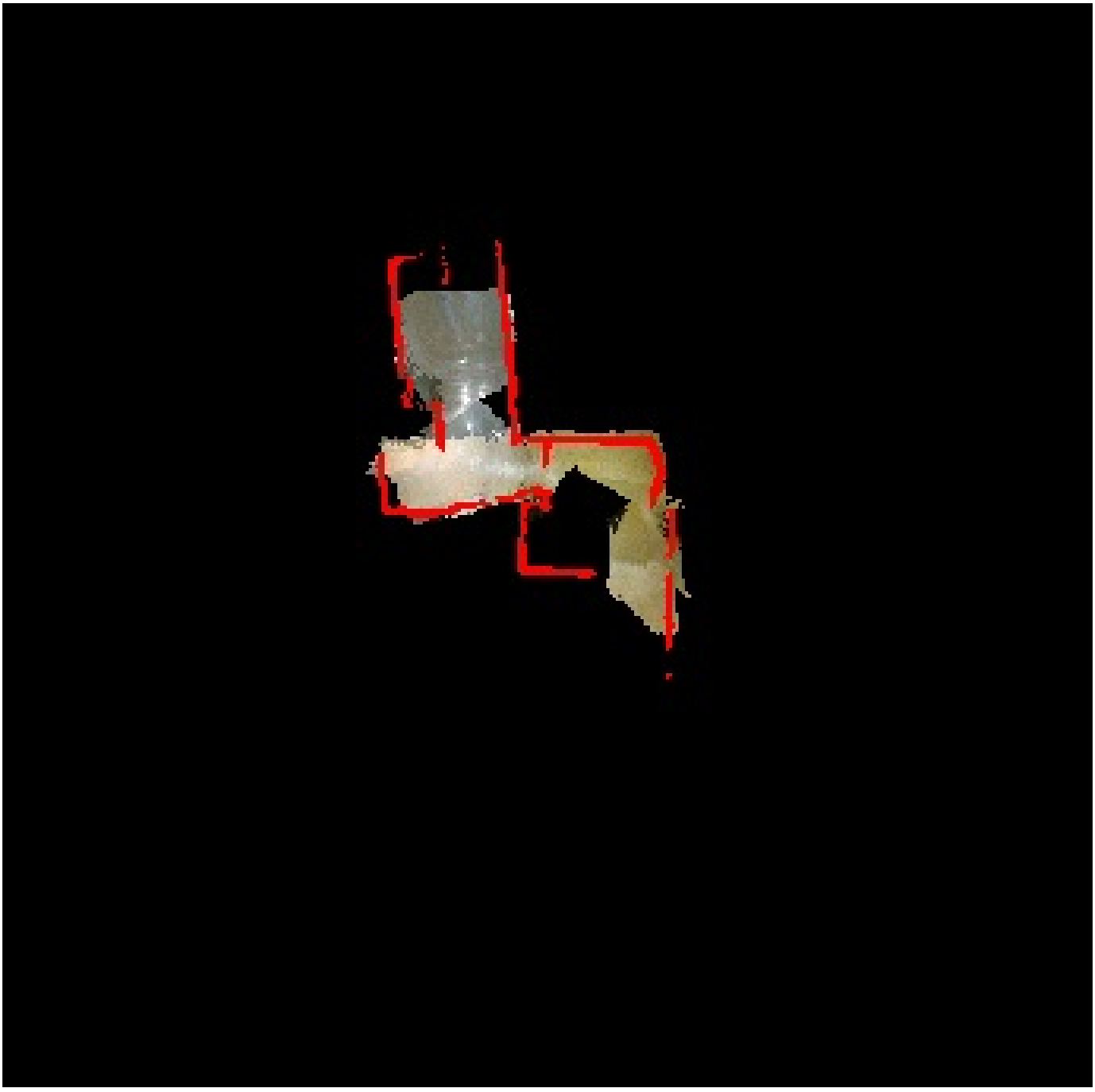

Fig. 13: Textured floor for Map3, red highlights show walls.

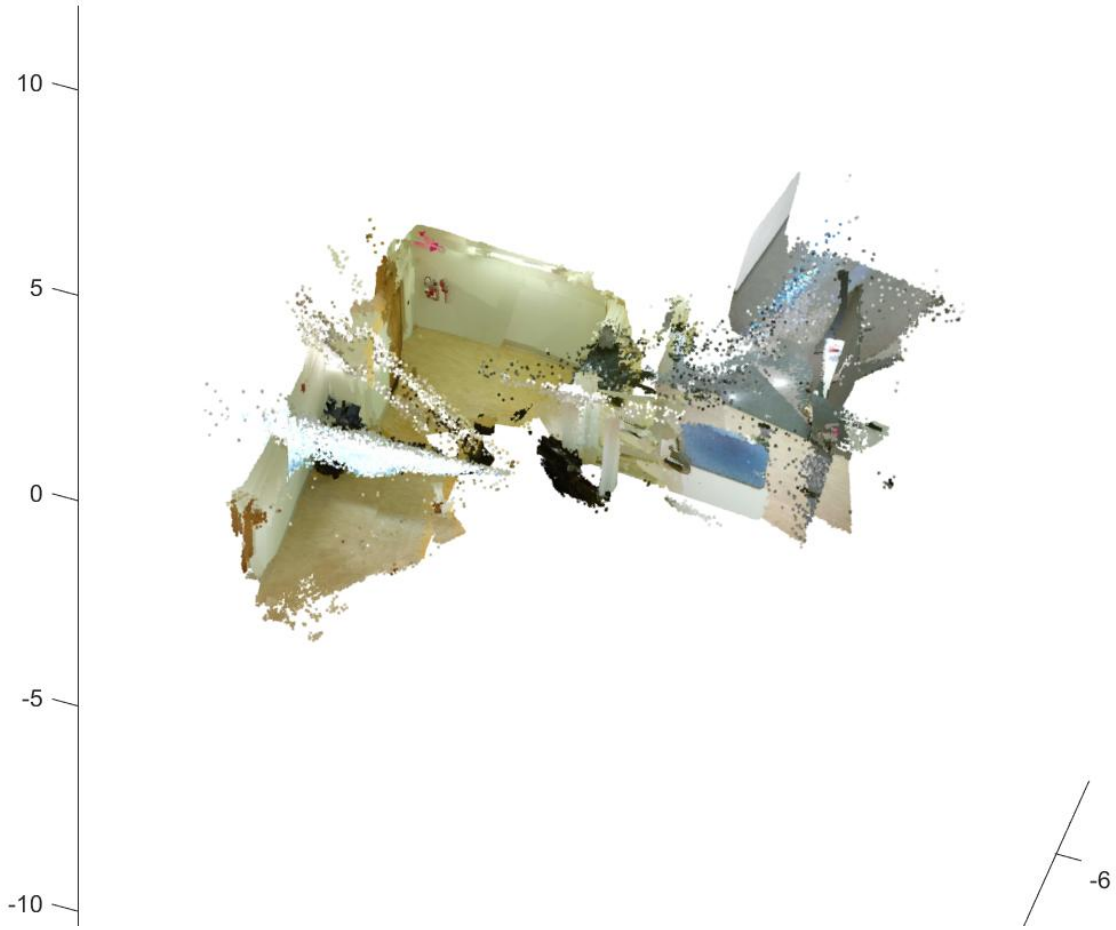

Fig. 14: 3D-map for Map3.

B. Test Set

The occupancy grid for Test Map is shown in Fig. 15 and the textured floor is shown in Fig. 16, the 3D-map for Test Map is shown in Fig. 17.

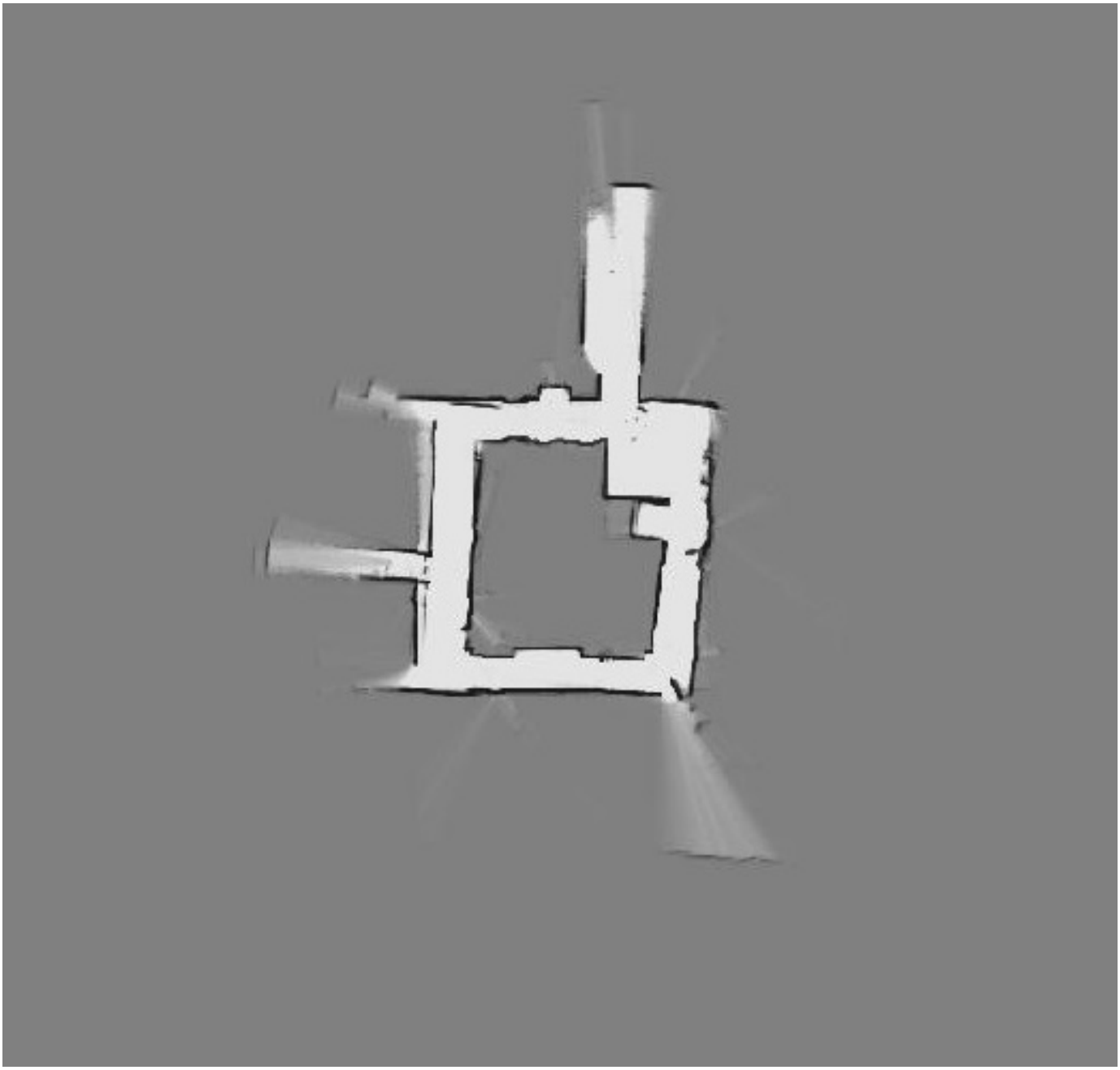

Fig. 15: Occupancy Grid for Test Map.

Fig. 16: Textured floor for Test Map, red highlights show walls.

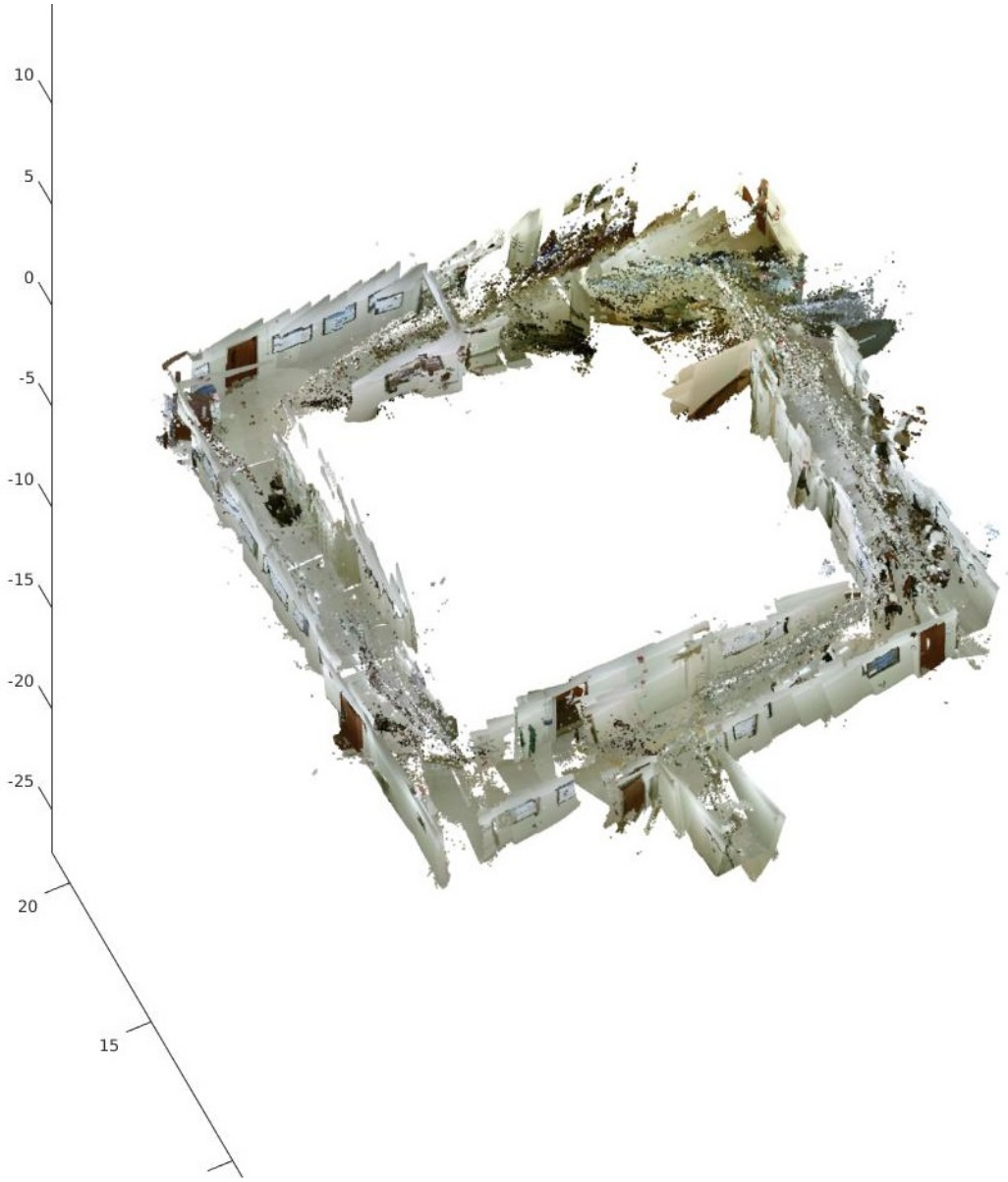

Fig. 17: 3D-map for Test Map.

XII. Important Lessons Learnt

Simplest algorithm is sometimes the best.
Optimizing RAM usage for speed.

XIII. Conclusions

In general, after the code was tuned for loop-closure, the same parameters worked very well with minimal drift for all the train and test datasets. Particle-filter based SLAM is robust and closes the loop very well. The 3D-RGB map and the floor overlay look good as well.

Acknowledgment

The author would like to thank Dr. Daniel D. Lee and all the Teaching Assistants of ESE 650 course for all the help in accomplishing this project.

References

[1] Frank Dellaert, et al. *Monte carlo localization for mobile robots*, IEEE International Conference on Robotics and Automation, Vol. 2., 1999.
[2] Sebastian Thrun, Wolfram Burgard, and Dieter Fox. Probabilistic robotics. MIT press, 2005.
[3] Peter's Matlab Toolbox, Link