Simultaneous Localization and Mapping on a Humanoid Robot

Alexander Khoury
A11563298
17 December 2017

I. INTRODUCTION

In many applications in robotics, in order for a robot to properly interact with the world around it, it requires information about its position in this world, and information about the world itself. This presents a problem in unexplored – in the robots perspective – environments, as the robot is seeing the world for the first time, and somehow has to infer its position from this limited information. This paper presents a solution to this problem, achieving simultaneous localization and mapping.

II. PROBLEM FORMULATION

Using data from odometry, IMU, and lidar on a humanoid robot, implement simultaneous localization and mapping (SLAM). Maintain a 2-D map of the environment, and the robots trajectory. Use RGBD data to color the floor of the 2-D map.

- **Mapping**:
  Using a laser scan and the current robot position, create/update a 2-D occupancy grid with the information from the scan.
- **Localization**:
  Using a particle filter, estimate the pose of the robot over time. Using a prediction update sequence, predict the next position of the particle using the odometry measurements, and update this position using scan matching on the current map.
- **Texture mapping**:
  Implement depth image to rgb image registration, then transform the points from the optical frame to the world frame. Threshold the height to determine the floor points, then paint the map floor the appropriate color determined by registration.

III. TECHNICAL APPROACH

A. The Robot: THOR

Thor is a humanoid robot equipped with many different sensors, of which this project only uses a few. Using the geometric properties of the robot itself, homogeneous transforms could be made from origin of the robot to the various sensors. The origin is defined as the torso of the robot.

$$_{origin}T_{sensor} = \begin{bmatrix} _{origin}R_{sensor} & _{origin}p_{sensor} \\ 0 & 1 \end{bmatrix} \quad (1)$$

Where $_{origin}R_{sensor}$ and $_{origin}p_{sensor}$ represent the rotational and translational offsets from the origin to the sensor. Oftentimes, smaller transforms can be chained to produce the final transform. For example to determine the transform from lidar (l) to origin (b) through the head frame (h):

$$_bT_l = \begin{bmatrix} _bR_h & _bp_h \\ 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} _hR_l & _hp_l \\ 0 & 1 \end{bmatrix} \quad (2)$$

Thor has a head motor and neck motor, allowing for the head to pitch and yaw. Throughout the SLAM algorithm, we must recalculate our transforms constantly to account for head movement.

The world frame is defined at $[0, 0, 0]$ and is initialized on robot startup. As the robot moves, the robot pose is defined in terms of this frame.

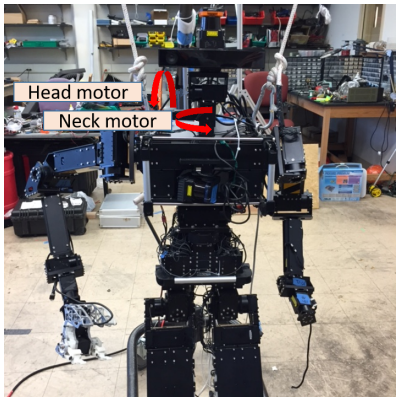

Fig. 1. THOR

B. Mapping

Given the robot pose, and the lidar scan measured at that time, the goal is to construct a 2-D occupancy grid denoting free, occupied, and unexplored spaces.

1) Remove scan points that are too far $r > 30\ meters$ or too close $r < 0.2\ meters$
2) Transform lidar scan from Polar (r, θ) to Cartesian (x_l, y_l, z_l) in lidar frame (l):

$$x_l = r \cdot \cos(\theta) \quad y_l = r \cdot \sin(\theta) \quad z_l = 0 \quad (3)$$

3) Transform lidar points (x_l, y_l, z_l) from lidar frame (l) to the world frame (w):

$$\begin{bmatrix} x_w \\ y_w \\ z_w \\ 1 \end{bmatrix} = {}_wT_l \cdot \begin{bmatrix} x_l \\ y_l \\ z_l \\ 1 \end{bmatrix} \quad (4)$$

${}_wT_l$ as described in the previous section by chaining transforms from lidar to head to origin to world.

4) Threshold z_w to remove floor points. Due to the fact the head is able to pitch, when the robot looks downwards, the lidar will see the floor, and we do not want to treat them as objects/walls. Therefore we remove points $z_w < .1$.

5) In order to visualize this map, the points were scaled and translated such that they can be mapped on an image plane. Using an map resolution of .05 meters per pixel, each point is transformed accordingly. The world frame is shifted to the center of the image. Due to the fact we are looking to create a 2-D map, z_w becomes unimportant, and only x_w, y_w need to be converted to coordinates in the occupancy grid x_c, y_c.

6) These coordinates represent locations where the lidar saw an object/wall, which corresponds to an occupancy. Thus because the lidar's ray was able to reach that coordinate, we can conclude that all the points leading up to that position from the robot pose are unoccupied or free. Bresenham's 2D ray-tracing algorithm allows for the calculation of these free points by tracing from the robot pose to the occupied cell.

7) With a collection of occupied coordinates and free coordinates, the occupancy grid can be updated. For each cell in the occupancy grid is an associated probability for which that cell is occupied. These probabilities can be accumulated per iteration, and thresholded to create the occupancy map. For example, we can threshold the probability map as:

$$\begin{aligned} free\ coordinates &= probability\ map < 0.3 \\ occupied\ coordinates &= probability\ map > 0.7 \end{aligned} \tag{5}$$

These thresholds can be adjusted to suit performance and quality needs.

8) In order to update the probability map, we must accumulate the odds that a certain cell is occupied/free. We define a map called the log odds map, where these odds are accumulated. The odds of a measured lidar scan can be computed from a ratio of trust in this measurement. The odds of any given cell m_i being occupied $g(1)$, can be computed as:

$$g(1) = \frac{P(z_t = 1 | m_i = 1)}{P(z_t = 1 | m_i = 0)} \tag{6}$$

Where z_t represents whether the lidar scan indicated the cell as occupied at time t. The odds of the cell being free $g(0)$ is subsequently computed as:

$$g(0) = \frac{1}{g(1)} \tag{7}$$

Thus the log odds map can be updated by accumulating the $\log(odds)$ at the current time step for each cell.

$$\lambda(m_i)_t = \lambda(m_i)_{t-1} + \log(g(z_t)) \tag{8}$$

where $\lambda(m_i)_{t-1}$ represents the accumulated log odds of the previous time step and z_t represents whether the lidar measured the cell occupied (1) or free (0) at the current time step t.

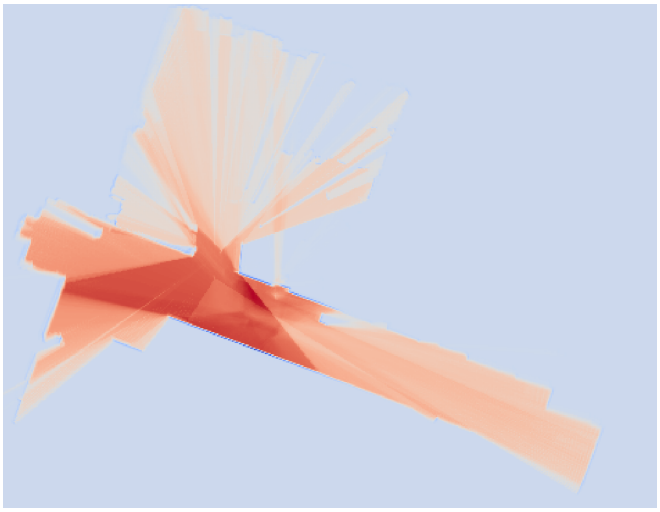

Fig. 2. Log odds map for Dataset 0

Although hard to see, the larger the odds in a cell being occupied, the more blue it is. The larger the odds in a cell being free, the more red. The odds of unexplored cells remains 0, and it is shown in light blue.

The probability of the cell being occupied can be recovered:

$$p(m_i | z_{0:t}) = 1 - \frac{1}{1 + e^{\lambda(m_i)_t}} \tag{9}$$

And subsequently the occupancy grid can be calculated using (5).

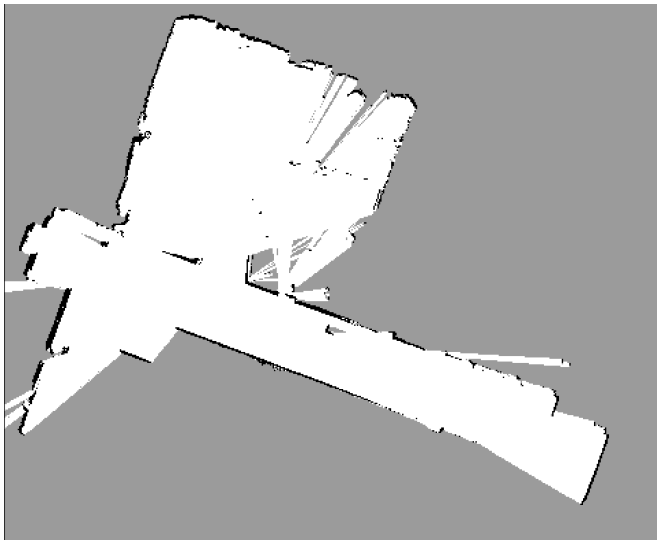

Fig. 3. Occupancy grid for Dataset 0

C. Localization

In order to localize the robot in the environment, we can implement a particle filter. One way to look at the particle filter is that stores, perturbs, and updates a number of hypotheses for the position of the robot. More formally, it is approximating the distribution of the state at time t given observations up to time t, denoted as $x_{t|t}$, as a weighted sum of delta functions.

$$x_{t|t} = [\mathrm{x}_t, \mathrm{y}_t, \mathrm{z}_t, \theta_t]^T, \quad p_{t|t}(x) \sim \sum_{k=1}^{K} \alpha_{t|t}^{(k)} \cdot \delta(x_{t|t}; \mu_{t|t}^{(k)}) \tag{10}$$

$x_{t|t}$ represents the 3-D pose of the robot, although in our case, $\mathrm{z}_t = 0$. K represents the number of particles that are initialized.

Initialization:
In order to initialize our particles, K random vectors $\mathcal{R}^4$ can be sampled from a Gaussian distribution, defining K perturbations $v^{(k)}$:

$$\begin{aligned} &v^{(k)} \ where \ v^{(k)} \ is \ sampled \ from \ \mathcal{N}(0, V) \\ &V = \mathrm{diag}(\sigma_x^2, \sigma_y^2, 0, \sigma_\theta^2) \end{aligned} \tag{11}$$

Note that the variance associated to the z component of the pose is 0, because we are only dealing with a 2-D map, and thus perturbations to the z component would be unnecessary.

Also note that our pose $x_{t|t}$, can be written in terms of homogeneous transforms $X_{t|t}$:

$$X_{t|t} = \begin{bmatrix} R(\theta_t) & p_t \\ 0 & 1 \end{bmatrix} \tag{12}$$

Where $R(\theta_t)$ represents the rotation matrix generated by θ_t. And p_t represents the position components of the state, $[\mathrm{x}_t, \mathrm{y}_t, \mathrm{z}_t]^T$ Similarly, our noise vector can also be written in this form as $V_{t|t}$. Thus, our particles can be initialized with the product of these transforms.

$$X_{0|0}^{(k)} = X_{0|0} \cdot V_{0|0}^{(k)}$$

Where we let $x_{0|0} = [0, 0, 0, 0]$.
Our weights $\alpha_{t|t}^{(k)}$ can all be initialized to $1/K$, giving equal probability to each particle.

Motion Model and Prediction:
Thor is moving throughout the environment, thus these particles need to be propagated through the model that describes the motion of Thor. Note that our control input $u_{t|t}$, which describes the movement of the robot, can be written in terms of homogeneous transform as well, $U_{t|t}$. Thus at every time step, the prediction step of the particle filter consists of perturbing all of the particles, then applying the control input transform $U_{t|t}$ to each particle. Thus our motion model becomes:

$$X_{t+1|t}^{(k)} = X_{t|t}^{(k)} \cdot V_{t|t}^{(k)} \cdot U_{t|t} \tag{13}$$

In order to determine our control input, we utilize the odometry data. Note that the odometry data suffers from a drift over longer periods of time, thus we define the control input $u_{t|t}$ as the difference between two consecutive odom poses, and since the time difference between the poses is small, this drift is minimized.

$$u_t = o_{t+1} - o_t$$

However, our control inputs and odom poses live in SE(3). Thus we must use a similar method as our motion model, representing each odom pose as a transform, with subtraction being defined in this space as an inverse:

$$U_{t|t} = O_{t+1}^{-1} \cdot O_t \tag{14}$$

Observation Model and Update:
In order to update the prediction step, we define a observation model very specific to our application. Our observation model is defined as a laser scan matching model, which requires knowledge of our current occupancy map m, laser scan z, and robot pose $x_{t+1|t}$, where we model the correlation between the scan and the current map.

1) Transform lidar scan to the world frame.
2) Compute the correlation between the map and the scan

$$\mathrm{corr}(z, m) = \sum_i \mathbf{1}\{z_i = m_i\} \tag{15}$$

which corresponds to the number of "hits" the scan makes on the map.

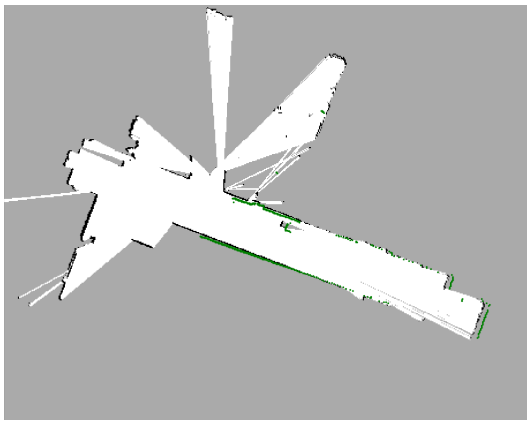

Fig. 4. Scan with high correlation

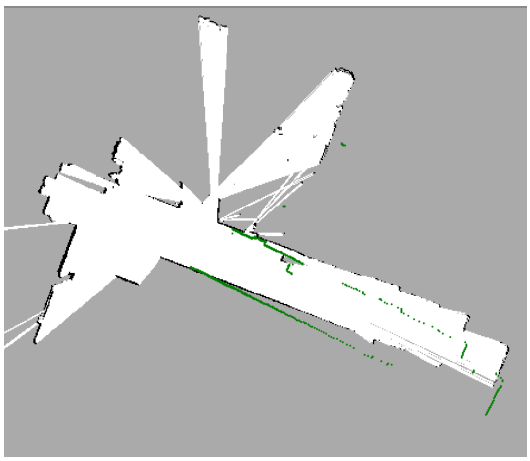

Fig. 5. Scan with low correlation

3) Use the softmax function to convert the correlation scores of all the particles to probabilities

$$P(z^{(j)}|x^{(j)}_{t+1|t}, m_t) = \frac{e^{\text{corr}(z^{(j)}, m_t)}}{\sum_k^K e^{\text{corr}(z^{(k)}, m_t)}} \tag{16}$$

4) The best particle is chosen as the particle with the highest correlation score, or largest $P(z^{(j)}|x^{(k)}_{t+1|t}, m_t)$.

5) The updated weights $\alpha^{(k)}_{t+1|t+1}$ can be calculated using the probabilities from the softmax function

$$\alpha^{(k)}_{t+1|t+1} = \frac{\alpha^{(k)}_{t|t} \cdot P(z^{(j)}|x^{(j)}_{t+1|t}, m_t)}{\sum_k^K \alpha^{(k)}_{t|t} \cdot P(z^{(k)}|x^{(k)}_{t+1|t}, m_t)} \tag{17}$$

6) However, if many of these correlations are bad, then we may get a lot of particles with very small weights ~ 0. In this case we may need to resample our particles to ensure a good number of quality hypothesis. In order to determine whether or not we need to resample, we compute the number of effective particles

$$N_{eff} = \frac{1}{\sum_k^K (\alpha^{(k)}_{t+1|t+1})^2} \tag{18}$$

If this value is less than a set threshold, for example $N_{thresh} = 0.1 * K$, then we resample.

7) A simple way to resample, is to just randomly select K new particles from the current particles with probability equal to their weight. The problem with this is there is chance the best particles (ones with high weight) will not be selected, thus losing a good hypothesis. We can improve upon this with a different resampling method. The method used for resampling is **Stratified Resampling**. This resampling method guarantees that samples with large weights appear at least once and those with small weights appear at most once. The resampling method creates K new particles, with new weights being identical $1/K$. These particles become the particles for the next iteration.

D. Texture Mapping

In order to color the floor of the map, the RGBD camera is used. The process for mapping the points is as follows:

1) Utilizing the transform between the infra-red camera and the RGB camera, and the intrinsics of both cameras, registration between the depth points and the colors of the RGB images can be computed.
2) Once the depth points have been matched to colors in the RGB image, we need to transform the depth points to the world frame in order to find the floor points.
3) Threshold the height of the world frame depth points, and determine what points are floor.
4) Paint the 2D map with the corresponding colors from the floor points.

IV. Results and Discussion

The results on the training data performed pretty well, with the exception of dataset 1, where midway through the scan matching was not able to extract and perpendicular features, meaning it kept thinking it was not moving, since it was matching vertical lines to vertical lines. The test dataset is similar to this, but I instead encountered a problem where the map would have a yaw offset at one of the corners which would mess up the scan matching the future when it closed the loop. In the time allotted, I was not successful in getting a good texture map computed. Below is the texture map for dataset 0.

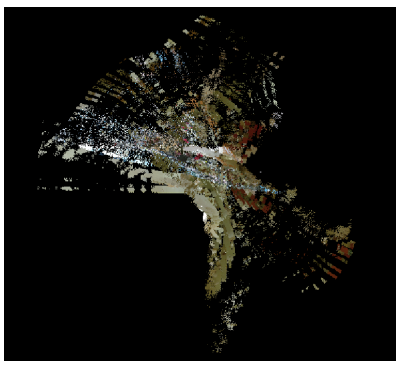

Fig. 6. Failed texture map for Dataset 0

Results on training data:

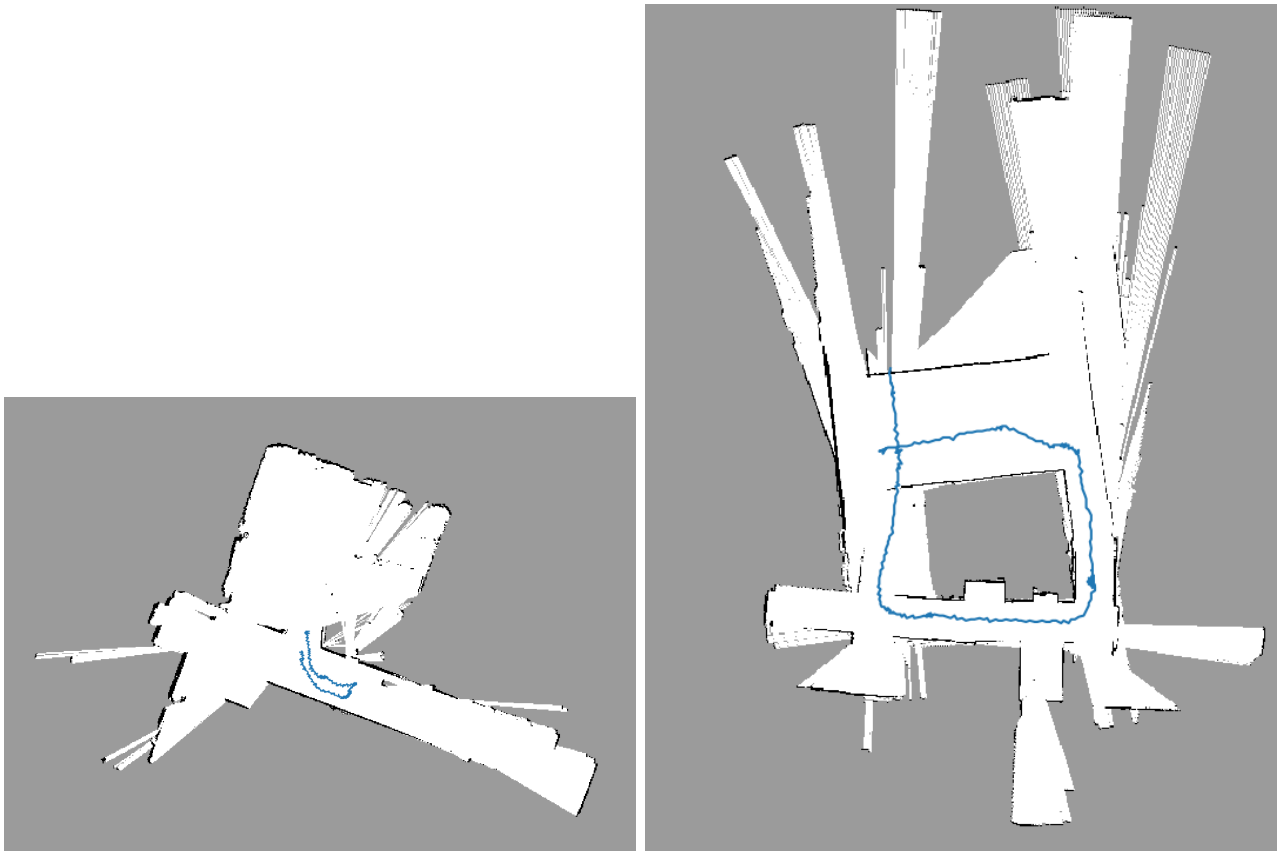

Fig. 7. Dataset 0 (Left) and Dataset 1 (Right). Note in the bottom right turn in dataset 1, there is pose ”ball” where the trajectory got stuck, if it had not gotten stuck there, the map would have looked really good.

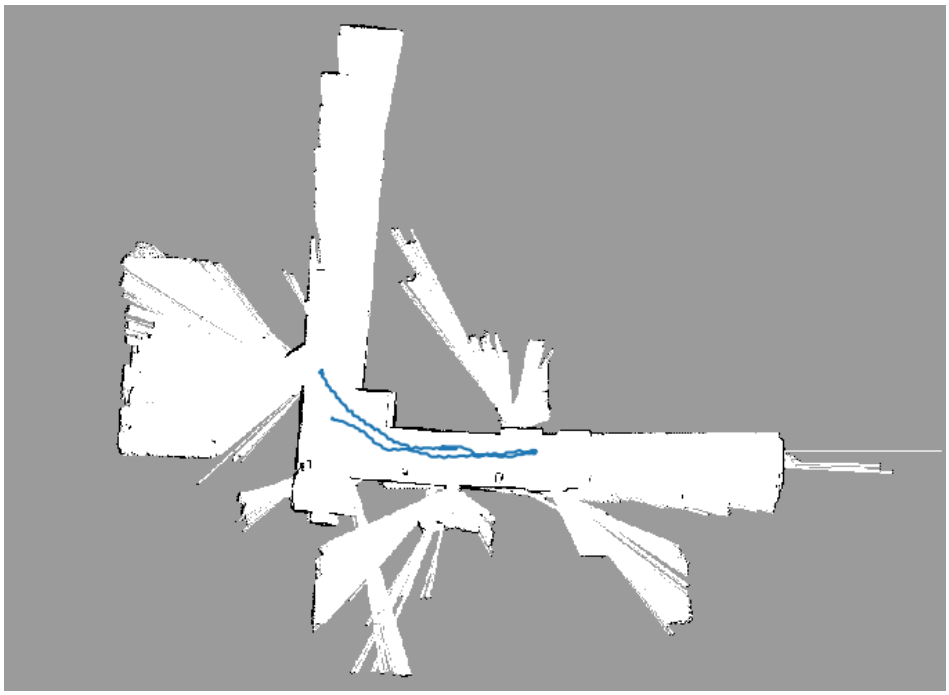

Fig. 8. Dataset 2

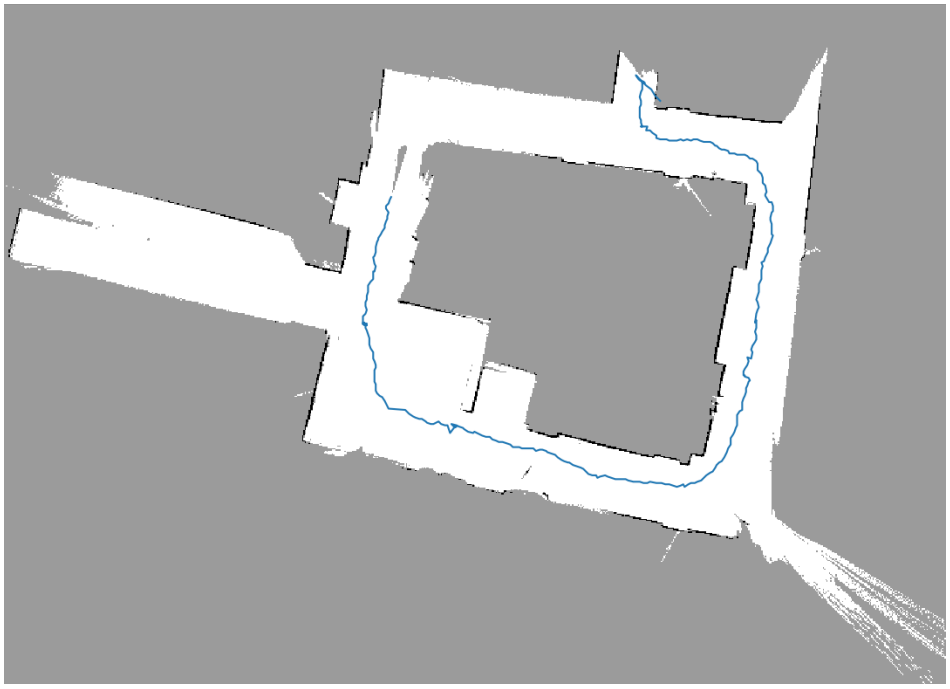

Fig. 9. Test Dataset. Note that after this point, it has a trouble matching the scan to close the loop.