Particle Filter for 2-D Simultaneous Localization and Mapping

Allen Zeng
UCSD
azeng@ucsd.edu

Abstract

This paper considers the use of a particle filter to solve the Simultaneous Localization and Mapping (SLAM) problem on a vehicle driving dataset. The dataset consists of sensor measurements from certain sensors mounted on the moving vehicle: a forward-facing 2-D LiDAR scanner, a fiber optic gyro, and wheel encoders. The odometry and LiDAR measurements are incorporated into the particle filter, using a discrete-time differential-drive kinematic model, to localize a robot while building a 2-D occupancy grid map of the environment. Finally, images from the vehicle-mounted stereo camera pair is used to texture the occupancy grid map generated in solving the SLAM problem.

1 Introduction

Autonomous vehicles (AVs) developed in the near future offer many benefits to society, one of which is providing a taxi or delivery service between locations in populated areas. These AVs must be able to navigate through changing environments. Before AVs can autonomously plan and navigate through an environment, an issue that must be addressed is that the AVs need to understand very the environment that they are driving through. Simultaneous Localization and Mapping (SLAM) can be performed to simultaneously build a map of the environment and localize the vehicle within the environment over time.

In this report, the 2-D SLAM problem for vehicles is addressed. Our dataset [1] consists of sensor measurements from a forward-facing 2-D LiDAR scanner, a fiber optic gyro (FOG), and two wheel encoders mounted on a vehicle driving through an urban area. A discrete-time differential-drive kinematic model and particle filter is used to solve this SLAM problem. The prediction step incorporates the FOG and encoder data, while the update step uses the LiDAR data. The vehicle environment is characterized by a 2-D occupancy grid map, and is filled out using the LiDAR information. Finally, once the complete environment and trajectory of the vehicle has been determined, the environment is colored-in using the vehicle's stereo camera information.

2 Problem Formulation

We are seeking to solve the 2-D Simultaneous Localization and Mapping (SLAM) problem in the case of a vehicle driving through an urban environment. Using the notation of [4], SLAM is a parameter estimation problem that attempts to solve for the robot state $\mathbf{x}_{0:T} \in \mathbb{R}^{3\times(T+1)}$ and environment map $\mathbf{m} \in \mathbb{R}^{m\times n}$ given a dataset of robot inputs $\mathbf{u}_{0:T-1} \in \mathbb{R}^{2\times T}$ and observations $\mathbf{z}_{0:T} \in \mathbb{R}^{r\times(T+1)}$. The goal is to compute:

$$p(\mathbf{x}_{0:T}, \mathbf{m} \mid \mathbf{z}_{0:T}, \mathbf{u}_{0:T-1})$$

where m and n are the number of occupancy grid cells in each dimension sufficiently large to characterize the environment, r is the number of data points per observation, and T is total timesteps.

The state $\mathbf{x}_t = [x, y, \theta]^\top$ characterizes 2-D locations x and y, and vehicle angle θ. Control inputs $\mathbf{u}_{0:T-1}$ can be inferred using the sensor data [3]. Here, $\mathbf{u}_t = [v, \omega]^\top$ where v and ω is linear and angular velocity respectively.

In the mapping step, the vehicle state trajectory $\mathbf{x}_{0:T}$ and measurements $\mathbf{z}_{0:T}$ are used to build map $\mathbf{m}$. In the localization step, the map $\mathbf{m}$, control inputs $\mathbf{u}_{0:T-1}$, and measurements $\mathbf{z}_{0:T}$ are used to infer the vehicle state trajectory $\mathbf{x}_{0:T}$.

After solving the SLAM problem, we texture the generated map using stereo camera information. Here we use the vehicle state trajectory $\mathbf{x}_{0:T}$, map $\mathbf{m}$, RGB stereo images $L_{0:T}, R_{0:T} \in \mathbb{R}^{h \times w \times 3}$, and corresponding camera projection matrices $P_L, P_R \in \mathbb{R}^{3\times4}$ to find the pixel colors corresponding to the plane of the 2-D occupancy grid. Problem: Find a function f that can identify that corresponding color map such that

$$f(\mathbf{x}_{0:T}, \mathbf{m}, L_{0:T}, R_{0:T}, P_L, P_R) = y \in \mathbb{R}^{m \times n \times 3}$$

3 Technical Approach

We utilize the particle filter approach to solve the stated SLAM problem. This consists of constructing an occupancy grid map, and utilizing a discrete-time differential-drive kinematic model to predict vehicle trajectory. In the following sub-sections, we address the dataset, the motion model, the occupancy grid map, and the particle filter as a whole. The last sub-section addresses the map texturing problem.

3.1 Dataset

A dataset consisting of 2-D LiDAR, fiber optic gyro (FOG), wheel encoder, and stereo camera measurements is used [1]. Additionally, we are provided the coordinate transforms for vehicle-to-LiDAR ${}_LT_V \in \mathbb{R}^{4\times4}$, vehicle-to-FOG ${}_FT_V \in \mathbb{R}^{4\times4}$, vehicle-to-left-stereo-camera ${}_CT_V \in \mathbb{R}^{4\times4}$. We chose the FOG coordinate frame to be body frame, such that ${}_BT_F = I_4 \in \mathbb{R}^{4\times4}$. These corresponding inverse transforms can be found by inverting the matrices. So, sensor-to-body transforms can be calculated by chaining together the appropriate transforms, for example LiDAR-to-body as

$${}_BT_L = {}_B T_F \cdot {}_F T_V \cdot {}_V T_L$$

. The 2-D LiDAR data provides range information over a 190° field-of-view over time. It is our observation variable $\mathbf{z}_{0:T} \in \mathbb{R}^{r\times(T+1)}$, where there are $r = 286$ per timestamp in this case. The LiDAR has a maximum range of 80 meters and infinite range observations are set as 0 value, so range data that is greater than 80 meters and less than 0.1 meters is discarded. The remaining ranges are converted to local Cartesian coordinates, given the range and angle information, and then the corresponding local homogeneous coordinate $[x, y, 0, 1]^\top$. These coordinates can be converted to the body frame by left-multiplying ${}_BT_L$. And in the particle filter update step, they are converted to world coordinates by left-multiplying the body-to-world transform ${}_WT_B$. These LiDAR points are used to denote occupied cells in the occupancy grid.

The FOG provides the delta roll, delta pitch, and delta yaw per timestamp. Since we are only addressing the 2-D SLAM problem, only the delta yaw information is used. The delta yaw $\Delta\theta = \tau\omega$ where ω is instantaneous angular velocity and τ is the time difference in the sample timestamps [1]. The wheel encoders provide an estimate of the instantaneous linear velocity per wheel. Knowing the encoder resolution r, wheel diameter d, and given the number of encoder ticks per timestamp z, we calculate [2]:

$$\tau v \approx \frac{\pi \cdot d \cdot z}{r}$$

where v is instantaneous linear velocity and τ is again the time difference in the sample timestamps. In our approach, the instantaneous linear velocity is estimated as the average of the velocities calculated from the left and right wheel encoders. The linear velocity is synchronized to the FOG timestamps by interpolating between encoder timestamps and assuming the change in linear velocity is constant between the timestamps. The v and ω are used in the motion model described in the next section.

Stereo camera images and parameters are also given in the dataset, and their use is described in the Texture Mapping sub-section below.

3.2 Motion Model

The discrete-time differential-drive kinematic model with Euler discretization [2] is used here. Because of the high frequency of the FOG data (and small yaw changes per timestamp), the exact discretization formula provided in [2] was empirically found to not make any significant different on the final SLAM result. The Euler discretization formula is repeated here. For time interval τ, the motion model f is,

$$\mathbf{x}_{t+1} = \begin{bmatrix} x_{t+1} \\ y_{t+1} \\ \theta_{t+1} \end{bmatrix} = f(\mathbf{x}_t, \mathbf{u}_t) := \mathbf{x}_t + \tau \begin{bmatrix} v_t \cos(\theta_t) \\ v_t \sin(\theta_t) \\ \omega_t \end{bmatrix}$$

This will be used in the prediction step of the particle filter.

3.3 Occupancy Grid

For this dataset, an occupancy grid is initialized with parameters such that the robot trajectory and corresponding LiDAR scans are all contained within the grid extents. Specifically, the parameters are chosen as $x_{min} = -100,\ x_{max} = 1500,\ y_{min} = -1400,\ y_{max} = 200,\ resolution = 1$, all in meters. The x and y extents are meters in world coordinates, and assumes the vehicle starts at the origin with identity rotation. (These values set based on the vehicle's trajectory calculated by dead-reckoning.) The resolution value indicates that each cell corresponds to one square meter of physical space. World coordinates (x_W, y_W) are converted to occupancy grid cell indices (u, v) by the formula

$$u = \left\lceil \frac{x_W - x_{min}}{resolution} \right\rceil - 1$$
$$v = \left\lceil \frac{y_W - y_{min}}{resolution} \right\rceil - 1$$

The LiDAR ranges, as mentioned in the Dataset sub-section, can be converted to world coordinates (x_W, y_W). Since the LiDAR ray ends at these coordinates, we know that there is an object there and so the corresponding occupancy grid cell should be marked as occupied. We also know the LiDAR ray traveled from the vehicle to that endpoint unobstructed, so the cells that the ray passes over is unoccupied. The cells that the rays pass through are determined using the 2D Bresenham line-drawing algorithm [1].

We maintain the occupancy grid as containing log-odds $\lambda_{i,t}$, where i is a cell index and t is time [4]. (The map is initialized with $\lambda_{i,0} = 0$.) For each cell observed as unobstructed, its value is updated as $\lambda_{i,t+1} = \lambda_{i,t} - \log(c)$. For occupied cells, $\lambda_{i,t+1} = \lambda_{i,t} + \log(c)$. Where, c models the "trust" in the sensor data as the ratio of true positives to false positives. It is assumed that the prior term is 1, meaning a cell is equally likely to be occupied or unoccupied. Consequently, the map probability mass function [4] $\gamma_{i,t}$ can be recovered using the sigmoid function:

$$\gamma_{i,t} = \frac{\exp(\lambda_{i,t+1})}{1 + \exp(\lambda_{i,t+1})}$$

Then, occupied cells have $\gamma_{i,t} > 0.5$ and unoccupied cells have $\gamma_{i,t} < 0.5$. Cells with $\gamma_{i,t} = 0.5$ are marked as unknown, as we have no information on their occupancy. A binary map can be produced from the log-odds map using this conversion. For map correlation function in the update step of the particle filter, the unknown cells are marked as free.

3.4 Particle Filter

We utilize the particle filter approach to solve the stated SLAM problem. There are four primary steps to applying the particle filter: first the initialization, then repeated prediction and update steps. Before every prediction step, particles are resampled as necessary. After the environment map is constructed and the vehicle trajectory is known, we traverse through the stereo images and color in the environment map. In the following paragraphs, the notation of [3] and [4] are followed.

The particle filter's particle set is initialized to zero, $\mu_{0|0}^{(k)} = 0 \in \mathbb{R}^3$, and weights to $\alpha_{0|0}^{(k)} = \frac{1}{N}$, where $N = 100$ is the number of particles chosen in our approach. The vehicle's pose is initialized

to the origin with identity rotation. Time $t = 0$ is set to the timestamp of the first available LiDAR scan, which is used to initialize the occupancy grid. Then, we perform resample, prediction and update steps as time advances. As noted in the dataset section, the encoder data is synchronized to the FOG timestamps. A resample (if necessary) and a prediction step is performed at every FOG timestamp. An update step, using the most recent predicted particles, is performed after the FOG timestamp closest to but right before a LiDAR timestamp. To speed up computation, we only perform a particle weight update and occupancy grid update every 5 LiDAR scans.

Before every prediction step, particles are resampled using Sample Importance Resampling (SIR) if the number of effective particles N_{eff} is less than some threshold number of particles (usually set as 20% of the total number of particles, as a rule-of-thumb). The number of effective particles is defined [3] as:

$$N_{eff} := \frac{1}{\sum_{k=1}^{N} \left(\alpha_{t|t}^{(k)}\right)^2}$$

When N_{eff} is less than the threshold, a new set of particles $\left\{\bar{\mu}_{t|t}^{(k)}, \bar{\alpha}_{t|t}^{(k)}\right\}$ for $k = 1, ..., N$. Using the notation of [3], we sample $j \in \{1, ..., N\}$ independently with replacement with probability $\alpha_{t|t}^{(j)}$. Then correspondingly add $\mu_{t|t}^{(j)}$ with weight $\frac{1}{N}$ to the new set of particles.

The particle filter prediction step is as follows, summarizing the ideas of [4]. For every particle $\mu_{t|t}^{(k)}$, $k = 1, ..., N$ representing the vehicle pose, we calculated the update as

$$\mu_{t+1|t}^{(k)} = f(\mu_{t|t}^{(k)}, \mathbf{u}_t + \epsilon_t)$$

where $f(\mathbf{x}, \mathbf{u})$ is the differential-drive motion model stated previously, $\mathbf{u}_t = (v, \omega)$ is the linear and angular velocity respectively, and ϵ_t is 2-D Gaussian motion noise. The noise is sampled by calculating the variance σ over the linear and angular velocity data,

$$\epsilon_t \sim \mathcal{N}\left(0, \begin{bmatrix} \sigma_v^2 & 0 \\ 0 & \sigma_\omega^2 \end{bmatrix}\right)$$

In the prediction step, the particle weights are unchanged.

The particle filter update step is as follows, again summarizing the ideas of [4]. The new particles weights are proportional to the previous particle weights:

$$\alpha_{t+1|t+1}^{(k)} \propto p_h(\mathbf{z}_{t+1}|\mu_{t+1|t}^{(k)}, \mathbf{m}) \cdot \alpha_{t+1|t}^{(k)}$$

where $p_h(\mathbf{z}|\mathbf{x}, \mathbf{m})$ is the LiDAR observation model and

$$p_h(\mathbf{z}_{t+1}|\mu_{t+1|t}^{(k)}, \mathbf{m}) = \text{softmax}\left(\textbf{corr}\left(\mathbf{y}_{t+1}^{(k)}, \mathbf{m}\right)\right)$$

Where $\textbf{corr}\left(\mathbf{y}_{t+1}^{(k)}, \mathbf{m}\right) = \sum_{y_{i,t}} 1\{y_{i,t} = m_i\}$ is *scan-grid correlation*. Here $y_i = r(\mathbf{z}_i, \mathbf{x}_i)$, denotes the lidar scan endpoints converted to world coordinates and then occupancy grid cell coordinates. The metric is maximized when the new LiDAR scan endpoints fall on top of the occupied cells in the map exactly. In this scan grid correlation, per particle pose, we search for the maximum correlation value while varying the world coordinates of the particle position over a range of $[0.4, 0.4]^2$ with resolution 0.1 meters. The coordinate shift corresponding to the maximum correlation value is also used to adjust the particle's (world coordinates) pose because the occupancy grid is limited to the 1 meter resolution. The new weights are normalized such that they sum to 1. Finally, the LiDAR scan corresponding to the particle with the largest weight is used to update the occupancy grid.

The resample, prediction, and update steps are repeated until the algorithm has processed all of the data. As a result, we have tracked the trajectory of the highest-weighted particle through time and have generated an occupancy grid map of the environment.

3.5 Texture Mapping

Stereo camera images and parameters are also given in the dataset. First, these images are debayered, and then the images are undistorted using the corresponding parameters. A disparity map is generated is then generated using OpenCV's Stereo Block-Matching algorithm, as shown in provided

example code [1]. The generated disparity map is converted to depths using the formula [2]:

$$d = \frac{1}{z} f s_u \cdot b$$

where d is a disparity value, z is depth, $f s_u$ is the camera horizontal focal length, and b is the stereo camera baseline. The plane that corresponds to the occupancy grid is found by thresholding the height of the 3-D coordinate that correspondingly projects into each pixel. The occupancy grid is then colored using the average color of the pixels that whose 3D points falls into each grid cell. This process is performed on the whole trajectory so that the whole occupancy grid is colored in.

4 Results

The implementation of the particle filter was done in several steps. First, the occupancy grid and its initialization using the first LiDAR scan was implemented. The result of this is plotted in Figure 1, which serves as a sanity-check of the coordinate transforms and mapping functions. Then the update step of the particle filter was implemented. To test the implementation initially only one particle is used and zero noise is added, such that dead-reckoning is performed. This is shown in Figure 2. The extents of the trajectory is used to inform the initialization of the full occupancy grid in the final algorithm implementation. In the next test, 50 particles is used and Gaussian noise is added. The result is shown in Figure 3. Finally, the prediction step of the algorithm is implemented. The result of running the particle filter, an occupancy grid and estimated vehicle trajectory, is shown in Figure 4. For this final plot, 100 particles was used. Additionally, I included `map.mp4` in submitting this project, which animates the occupancy grid and the vehicle trajectory over time. The particle filter works, and does not seem to significantly differ from the dead-reckoning trajectory. Since there is no ground-truth available, it is difficult to determine which trajectory is more accurate. However, the occupied cells match up well and there does not seem to be much unoccupied cells *behind* occupied cells. (E.g. free space behind walls.) This means the trajectory estimate is accurate enough to construct a good representation of the environment.

As part of the update step, it was suggested [1] that the particles' poses be refined using the coordinate-shifts corresponding to the best *scan-grid correlation*. I found that this did not work well in practice. When this technique is used, I found that the vehicle trajectory would drift backwards starting at the beginning and would only move forward when velocity is high enough. I believe this drift appears when the vehicle is moving less than 1 grid cell (1 meter) in between LiDAR scans. In this case, even though the vehicle has moved forward in world coordinates, the scan-grid correlation essentially states that the best match occurs location of the previous LiDAR scan. Thus adding the coordinate-shift refinement may push the vehicle backwards, causing bad results overall.

In the disparity map generated by Stereo Block-Matching, Figure 5, the disparity values are most accurate along well-defined edges, such as lane markers. For the road, the calculated disparity values are generally inaccurate because the road is mostly a uniform black color. This is shown by the noisy values on the road region. By only comparing small blocks along each image row, it is difficult to tell what correct disparity and the range to the ground should be. Stated another way, it is difficult to match features across images about visual texture-less regions. I was unable to produce a good-quality textured map of the environment because of this difficulty. The texturing result is shown in Figure 6.

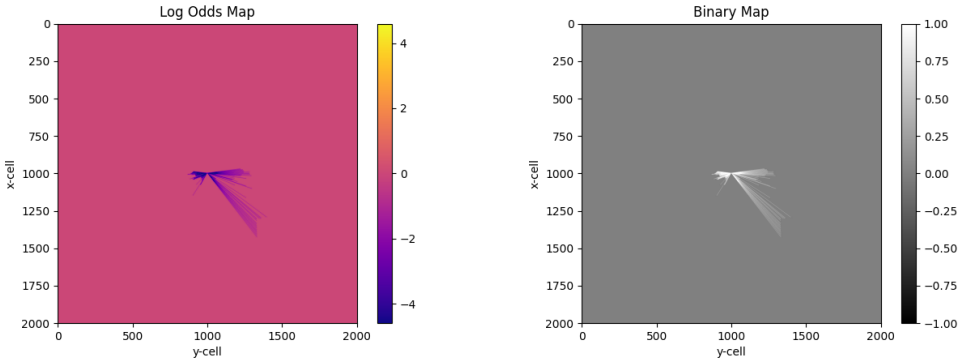

Figure 1: Log-odds map and binary map corresponding to the first LiDAR scan.

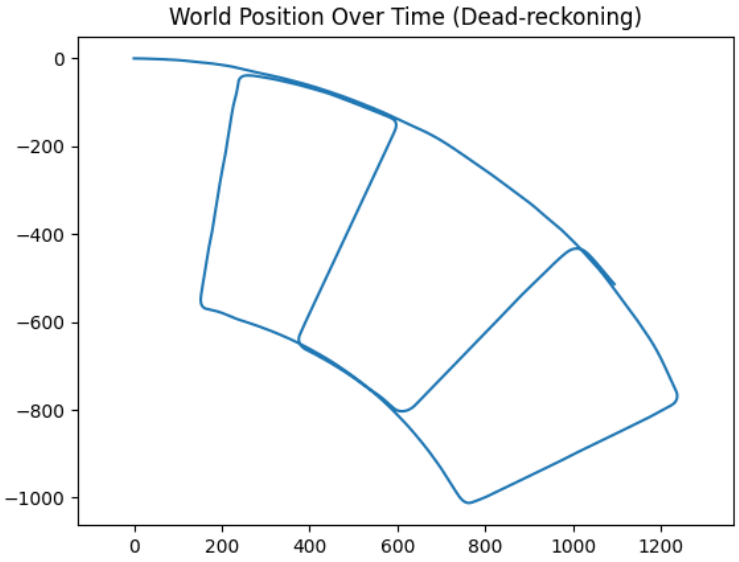

Figure 2: The vehicle trajectory calculated by dead-reckoning.

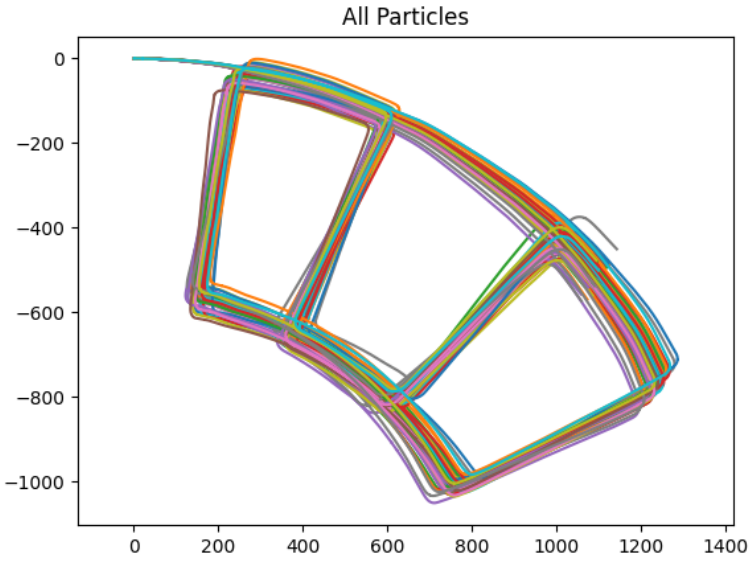

Figure 3: Plot of 50 particles' trajectory in world coordinates, performing only the prediction step.

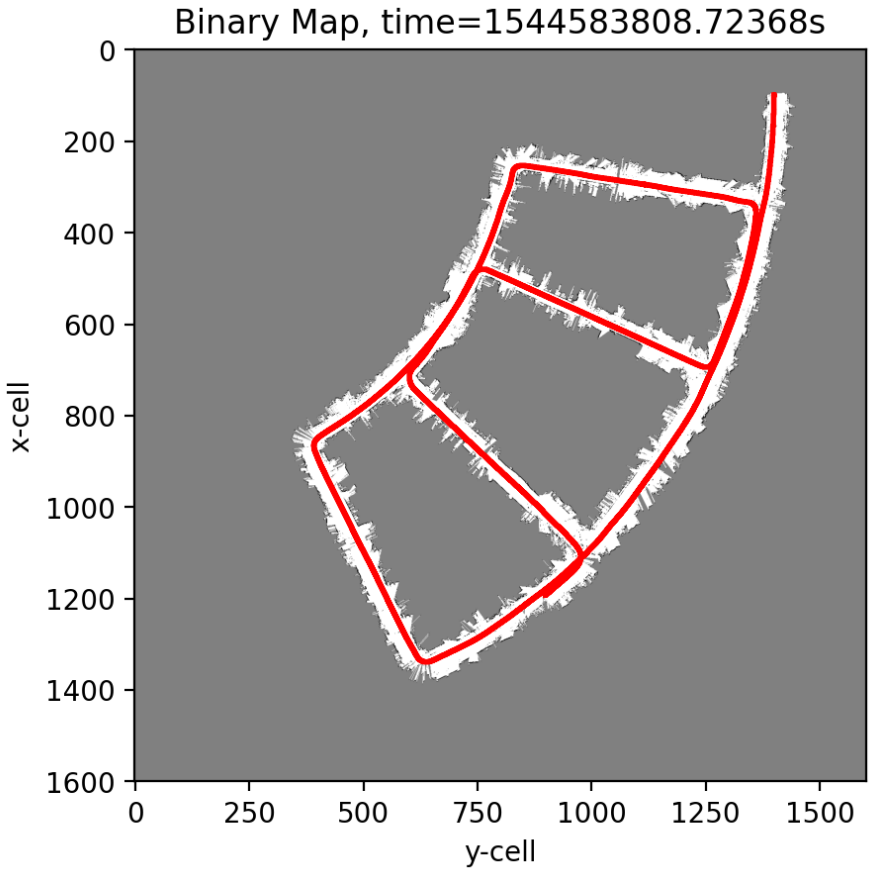

Figure 4: Occupancy grid, using every 5^{th} LiDAR scan, and 100 particles. The trajectory of the highest-weight particle is plotted in red. White pixels indicate unoccupied cells, black pixels indicate occupied cells, and gray pixels indicate unseen cells. The black pixels border the white regions, but are difficult to see because the occupied border is on the order of 1-pixel wide.

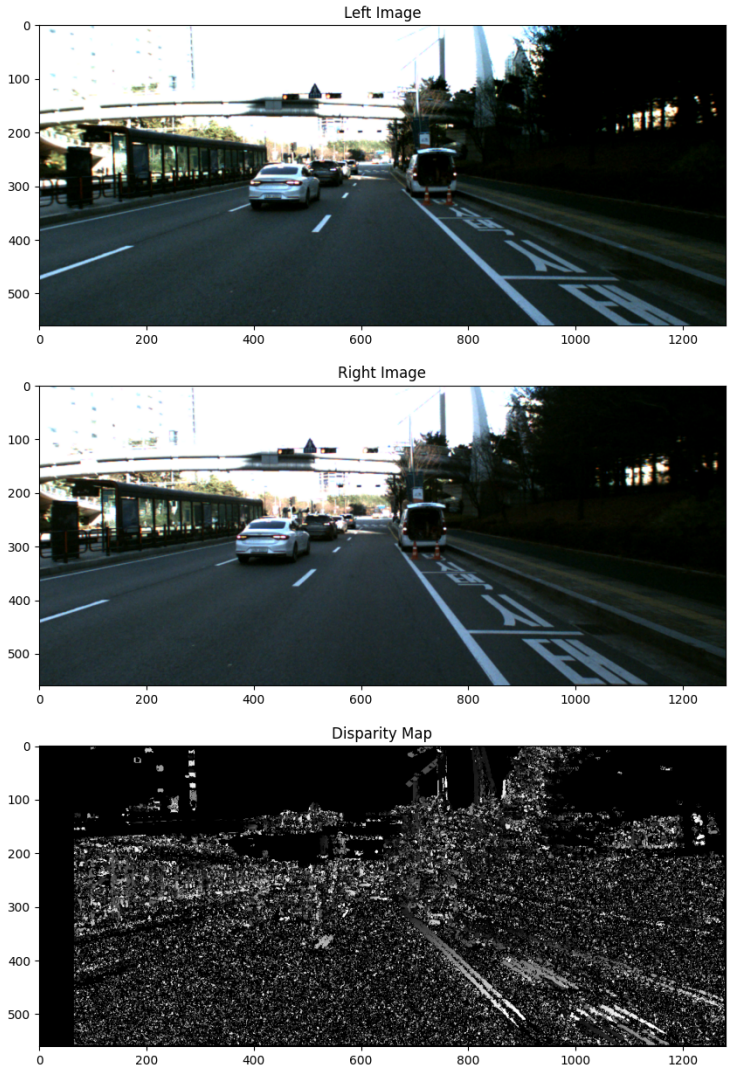

Figure 5: The first pair of stereo images and the corresponding disparity map. Brighter disparity pixels indicate higher disparity and therefore lower range from the camera. Ground disparities are ambiguous because much of the road is uniformly black.

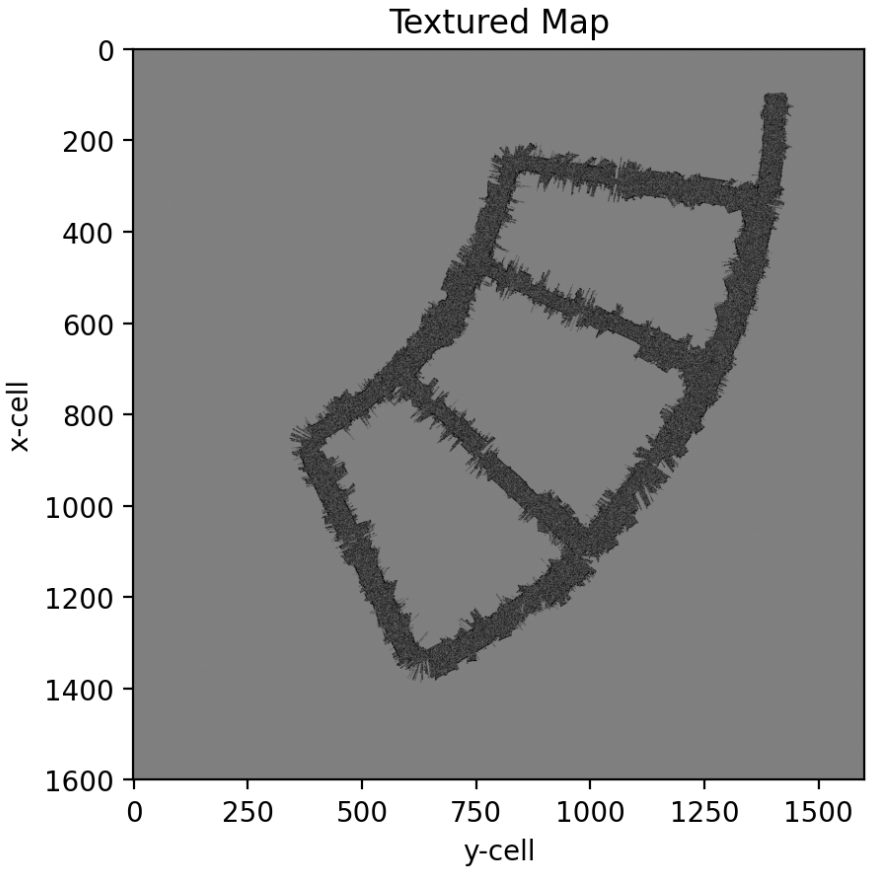

Figure 6: Map texturing result.

References

[1] Atanasov, N. (2021) *ECE276A: Sensing & Estimation in Robotics, Project 2: Particle Filter SLAM*. pp. 1-3

[2] Atanasov, N. (2021) *ECE276A: Sensing & Estimation in Robotics, Lecture 7: Logistic Regression*. pp. 1-41

[3] Atanasov, N. (2021) *ECE276A: Sensing & Estimation in Robotics, Lecture 8: Bayesian Filtering*. pp. 1-34

[4] Atanasov, N. (2021) *ECE276A: Sensing & Estimation in Robotics, Lecture 9: Particle Filter SLAM*. pp. 1-43