ESE 650 Homework 4 - SLAM with a Particle Filter

Tarik Tosun

March 25, 2014

Introduction

In this assignment, I implemented Simultaneous Localization and Mapping (SLAM) with a particle filter. The file step_slam.m steps the outer loop of my algorithm, and is summarized below:

```
function [pos, map, slam_state] = ...
    step_slam( data, slam_state, map, params )
```

(1) a priori estimate for particles using odometry

```
slam_state.particles = ...
    a_priori( slam_state, data, params );
```

(2) a posteriori estimate for particles and map using scan matching

```
[a_posteriori_weights, a_posteriori_map] = ...
    a_posteriori( slam_state, map, data, params );
```

(3) re-sample particles

```
slam_state = resample_particles( slam_state, params );
```

As with other Bayes filters, the particle filter has two main parts: the prior state estimate (propagating the system forward according to a motion model or transition model) and the posterior state estimate (which updates the state estimate to better align with the observations). In the sections following, I will explain the details of these two steps, and present my results on the training and test data sets.

Data, particles, and map

This project involved combining data from encoders, an IMU, a Hokuyo LIDAR, and a Kinect camera. I did not use the Kinect camera. A challenge in data processing was dealing with the fact that the sensor readings were not synchronized in time. I dealt with this by synchronizing all the sensor data to the Hokuyo times, by iterating over the Hokuyo samples and selecting the closest data point for each sensor to the Hokuyo sample. I used 20 particles for most data sets. Given more time, I would have vectorized my code and used more particles, but unfortunately without this optimization 20 particles was the most I could use and compute solutions in a reasonable amount of time. Resampling was implemented as presented in class, when the effective number of samples $N_{eff} = \frac{\left(\sum w_k\right)^2}{\sum w_k^2}$ was below $\alpha N_{samples}$. I used $\alpha = 0.5$ for the results presented here.

My map was an int8 array representing an area of 70mx70m at a resolution of 10cm. I used a prior of 0.85 for $p(laser\ says\ filled \,|\, cell\ actually\ filled)$ and 0.9 for $p(laser\ says\ empty \,|\, cell\ actually\ empty)$. In the code presented in the introduction, `slam_state`, `data`, and `params` are all structs. `slam_state` includes the particles, their weights, and the current time (for gyro integration). `data` includes all data fields for the current time step, including imu, encoders, and Hokuyo measurements. `params` includes all parameters of the particle filter; see `create_params.m` for the full list.

Prior estimate

`a_priori.m` propagates the particles forward according to the motion model, and introduces process noise to "jitter" the particles.

Odometry

Sensor data from the robot encoders and IMU were available for odometry estimation. Encoder odometry alone was not enough to get good results. When combined with the IMU for yaw estimation (through simple integration), odometry was fairly accurate. Plots of integrated odometry for the training set are shown in Figure 1. The mathematics of my odometry model are summarized below:

$$\Delta\theta = \omega_{yaw} * \Delta t$$

$$\Delta s = \frac{e_r + e_l}{2}$$

$$\Delta \begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} \cos\Delta\theta \\ \sin\Delta\theta \end{bmatrix} \Delta s$$

Where ω_{yaw} is the gyro yaw rate and e_r and e_l are the current left and right encoder values.

Process Noise

Process noise was modeled as gaussian random noise applied to the encoder and gyroscope readings (process noise was applied before the process model). In the results presented, I used a standard deviation of 5 for the encoders and 0.36 for the gyroscope.

Posterior estimate - LIDAR correlation

In the posterior update step, a Hokuyo-to-world transform is computed for each of the particles. These transforms are used to project the scan data onto the current map. The correlation of the putative scan for each particle with the current map is computed using the provided mex function `map_correlation`. We use this as a measure of confidence in that particle's prediction: if the scan projection of a particle aligns well with the current map, the particle probably provides a good estimate. Posterior weights are computed by adding the minimum of the particle scan-map correlations to each correlation value (ensuring that none will be negative), multiplying the prior weights by these correlation values, and rescaling to sum to 1. The map is then updated by adding the weighted sum of the transformed scans to the map, incrementing cells that are perceived as filled and decrementing those perceived as empty (using `getMapCellsFromRay.cpp` to determine empty cells).

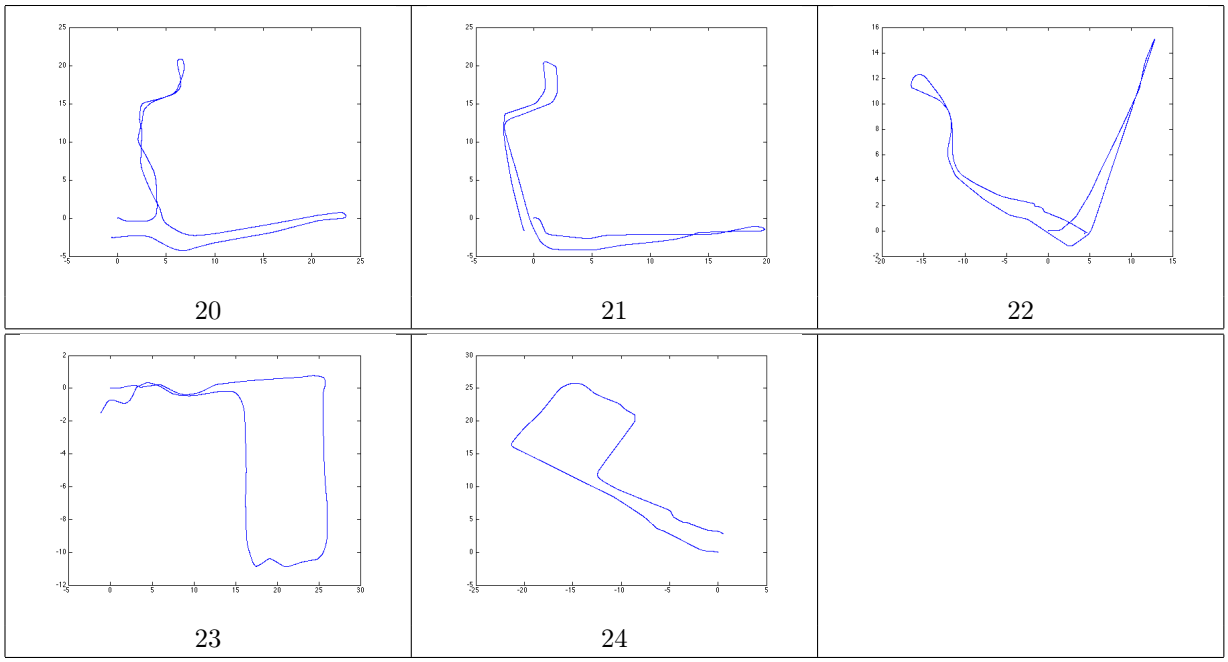


Figure 1: Pure Odometry Integration

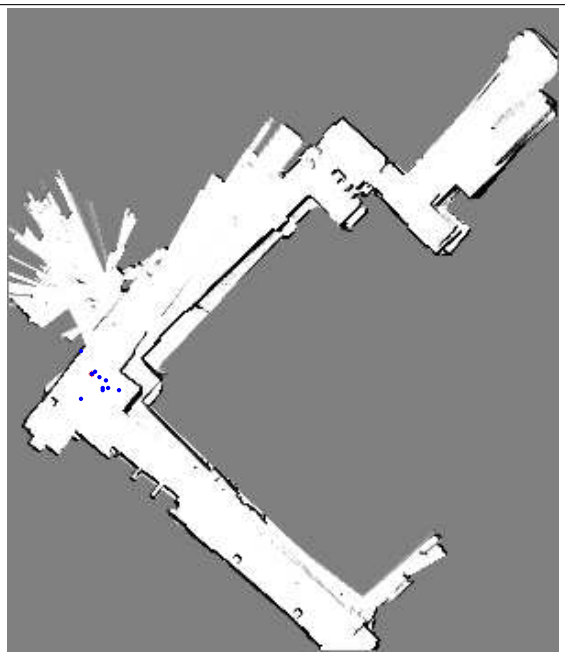
Results

Some maps from the training and testing set are shown in Figure 2.

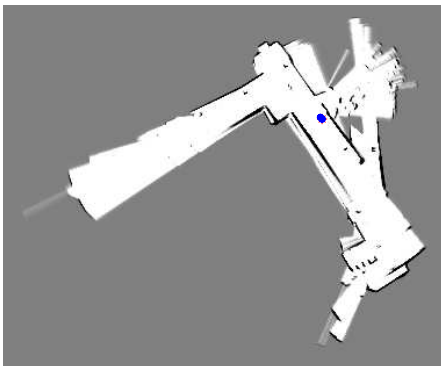
Analysis and Conclusion

Odometry integration alone was enough to produce good results for the training set, but for the test set it was necessary to close the loop with the particle filter to get good results (see Figure 3). On the test set, my filter performed well over short time periods, but often suffered from map shift over the course of a data set. As a result, we see curved hallways (test set 3), and sometimes bifurcations at points where the robot turned sharply and its orientation was estimated incorrectly (test set 1).

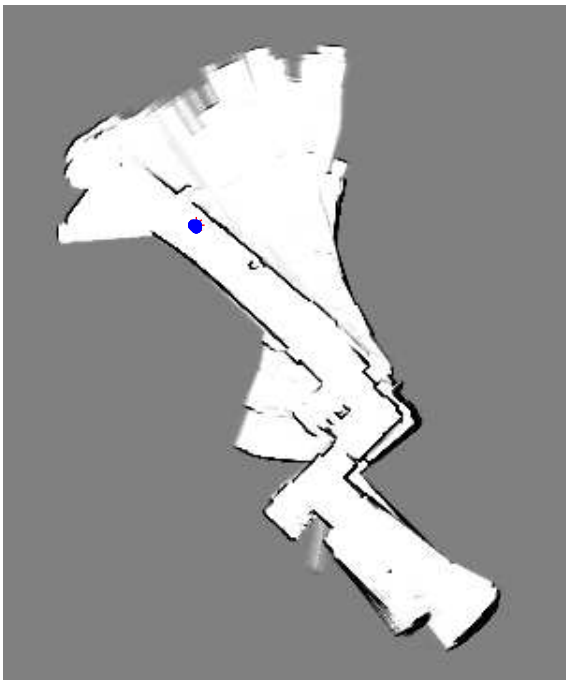
I found that my results depended heavily on my choice of parameters, including the laser scan priors, noise distributions, and number of particles. I found it challenging to tune these parameters in a methodical way, so tuning involved an extended period of trial-and-error. I think that if I had been able to use more particles (on the order of 100), performance would have been better and also more robust to parameter tweaks. I think my biggest mistake in this assignment was spending time tweaking parameters rather than vectorizing my particle computations, which could have allowed me to use many more particles.



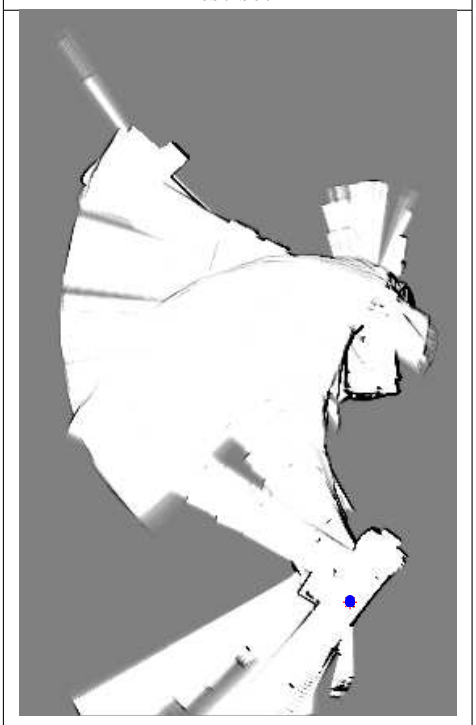
Training set 20



Test set 1



Test set 2



Test set 3

Figure 2: Results

Figure 3: Odometry integration on the test set produces post-modern surrealist mappings of GRW.