Particle Filter SLAM and Texture Mapping on an Autonomous Vehicle

Kwok Hung Ho
Department of Electrical and Computer Engineering
University of California, San Diego
La Jolla, U.S.A.
khh019@ucsd.edu

Nikolay Atanasov
Department of Electrical and Computer Engineering
University of California, San Diego
La Jolla, U.S.A.
natanasov@eng.ucsd.edu

I. Introduction

With the world moving into autonomous vehicles and robotics, the problem of Simultaneous Localization and Mapping (SLAM) has attracted many in the mobile robotics literature. This paper focuses on the SLAM via a particle filter on data collected from an autonomous car. By implementing a Fast-SLAM like approach, the pose of the car is estimated via posterior distributions of the K particles based on the weights assigned by log odds of a occupancy grid map. An optional texture map is also rendered alongside the particle filter through mapping the stereo cameras on board the vehicle onto the map. This implementation is also optimized to run in $\mathcal{O}(nK)$ time with n being the amount of data collected and K being the number of particles. As we see more and more autonomous vehicles in the modern world, reliable and fast SLAM systems are needed to ensure safety of passengers and civilians.

a differential drive motion model, and the observation model is realized via a 2D lidar. Through adding Gaussian noise to our motion model and correcting it based on the highest probability particle pose, localization despite motion error is achieved.

II. Problem Formulation

A. Problem Formulation: Data and Objective

Understanding that there can be many approaches to solve the problem, only the input, output, and associated models that cannot be changed and are inherent to the problem will be presented here. The technical approach, where solutions to the problems that will be described, will be presented in the next section.

The data provided includes timestamped readings from left and right wheel encoders, a fiber-optic gyroscope (FOG), a 190° FOV 2D lidar, and a pair of stereo cameras. Accompanying the data, parameters for each of the sensors including their transformations to the vehicle, encoder resolution, wheel diameters, lidar start and end angles, lidar resolution, lidar range, camera intrinsic matrices, and baseline are all given as well.

To be be more specific, all data except stereo camera images are given in CSV files. for the lidar data, the readings are given as [timestamp, $\frac{190}{0.666}$ readings as meters]. For the encoder, the data is given as [timestamp, left count, right count]. And for the FOG data, it is given as [timestamp, delta roll, delta pitch, delta yaw].

The timestamps of the different measurement data sources also have some offset, different sampling rates, and misalignment. More specifically, the FOG data is sampled ≈10X more than the encoder and lidar data, and ≈100X more than the stereo cameras, which only consisted of ≈ 1000 560×1280 stereo picture images for left and right cameras each.

With the data provided, the goal is to simultaneously map a 2D top down map of the world (free and occupied cells), and localize the robot's 2D pose (x, y, θ_{yaw}) onto the map, while also rendering a RGB texture map of the world onto a 2D map.

B. Problem Formulation: Motion and Observation Models

A motion model is defined as a function relating the current state **x** and control input **u** of a robot with its state change subject to motion noise **w**. It is parameters as follows:

$$\dot{x}(t) = f(x(t), u(t), w(t)) \tag{1}$$

$$x_{t+1} = f(x_t, u_t, w_t) \tag{2}$$

for continuous and discrete time respectively.

An observation model is a function relating the robot state x_t and the environment m_t with the sensor observation z_t subject to measurement noise v_t:

$$z_t = h(x_t, m_t, v_t) \tag{3}$$

with the following pdf due to the presence of measurement noise:

$$p_h(\cdot|x_t, m_t) \tag{4}$$

C. Problem Formulation: Texture Mapping

The goal of texture mapping is to color the top-down 2D world based on observations seen by the car from the stereo cameras. Given camera parameters, it is essential to use the extrinsic and intrinsic parameters to transform pixels to the camera frame, then to the vehicle frame, then to the 2D world frame. The approach to achieve this will be discussed in the next section.

D. Problem Formulation: Conclusion

With the problem formulated, the technical approach which describes how to solve the problem and the equations listed above will be presented, and one implementation, which is realized via a particle filter, will be shown in detail.

III. Technical Approach

A. Technical Approach: Introduction

The following technical approach will attempt to solve the problem formulated in the previous section via a particle filter, which is the centerpiece of this paper.

B. Technical Approach: Markov Assumptions

The SLAM problem via a particle filter, is best defined as a probabilistic Markov Chain since the current state is only dependent of the previous state and the previous control input. At time t, the robot's state x_t will be accompanied by a control input u_t and observation z_t. The map state m_t is assumed to only depend on m_{t-1}, but here is generalized to be largely static and independent of x_t.

C. Technical Approach: Motion and Observation Model

With the Markov Assumptions in place, the motion and observation models that are described by (2) and (4) can be defined. Since the robot state consists of (x, y, θ_{yaw}) only, a differential-drive kinematic model is used for motion.

$$x_{t+1} = \begin{bmatrix} x_{t+1} \\ y_{t+1} \\ \theta_{t+1} \end{bmatrix} = f(x_t, u_t) := x_t + \tau \begin{bmatrix} v_t cos(\theta_t) \\ v_t sin(\theta_t) \\ \omega_t \end{bmatrix} \tag{5}$$

For some time interval τ and where vt is the linear velocity, ω_t is the angular velocity at time t. This ensures that we only have to keep track of three variables: (x, y, θ_{yaw}), as linear velocity can be calculated from the encoder data, and θ_{yaw} can be read directly from FOG data. For linear velocity v_t, it has to be calculated via:

$$\tau v \approx \frac{\pi dz}{\text{ticks per revolution}} \tag{6}$$

For a time interval τ, given encoder count z, wheel diameter d. This is computed for each wheel and the linear velocity is computed as the average from both. With the difference in timestamps, $\Delta\theta_{yaw}$, and linear velocity v_t, (7) can be used to predict the car's pose.

For the observation model, lidar readings can be transformed into the vehicle frame. Since the readings are from -5° to 185° with resolution 0.666°, the lidar positions relative to the sensor can be calculated via simple geometry. For example, if the x and y axis are defined as right and forward to the car respectively, $r \cdot cos(\theta)$ and $r \cdot sin(\theta)$ is the x and y position of the lidar reading where r is the reading and θ is the angle in degrees $\in [-5, 185]$. After the lidar data is transformed into the sensor frame, it has to be transformed into the vehicle frame. To do this, the provided parameters: rotation matrix R and translation T are used.

$$s_V = Rs_s + T \tag{7}$$

Where s_V is the position in vehicle frame and s_s is the position in sensor frame. For simplicity, the lidar's z reading will be set to 0, hence making the 3by3 rotation matrix operation possible.

With the observation and motion model calculated, to model them as pdf's such as in (4), the particle filter and model will be implemented.

D. Technical Approach: Particle Filter

The particle filter is a Bayesian histogram filter with grid centers that move around to adaptively concentrate on certain areas in the state space. In the particle filter, a particle is defined as a hypothesis that the value of state x is $\mu^{(k)}$ with probability $a^{(k)}$, where k denotes the k^{th} particle. Therefore, $\mu^{(k)}$ represents the location, and $a^{(k)}$ represents the weight of the k^{th} particle. With $\mu^{(k)}$ and $a^{(k)}$ defined, the following probability density function's can be defined:

$$P_{t|t}(x_t) = \sum_{k=1}^{N_{t|t}} \alpha_{t|t}^{(k)} \delta(x_t - \mu_{t|t}^{(k)}) \tag{8}$$

$$P_{t+1|t}(x_{t+1}) = \sum_{k=1}^{N_{t+1|t}} \alpha_{t+1|t}^{(k)} \delta(x_{t+1} - \mu_{t+1|t}^{(k)}) \tag{9}$$

Where δ is the dirac delta function and N is the amount of particles at time t.

In the implementation, the particle filter begins the prediction of the robot's state at time $t = 0$ by defining the world frame coordinates $(0, 0)$ to be the same as the car, the lidar data is also read at $t = 0$, and points are plotted in a grid occupancy map with 1 indicating occupied points, and 0 indicating free space. The weights of all K particles are also initialized as $1/K$ in a $Kx1$ vector. Then looping over the encoder data, lidar data, and FOG data which, is accumulated for each 10 samples, the prediction of the car's pose is estimated. Next random Gaussian noise is added to this pose according to the number of particles to obtain a K by 3 matrix.

$$s_V = Rs_s + T \tag{10}$$

For each of the particles, the lidar points in vehicle s_V frame are transformed into the lidar points in world frame s_W via (9) where R is now the standard 2D rotation matrix:

$$\begin{bmatrix} R = cos(\theta) & -sin(\theta) \\ sin(\theta) & cos(\theta) \end{bmatrix} \tag{11}$$

and T is a 2D vector indicating the current particle's x and y position. For each particle, its respective lidar points are passed into a laser correlation model described by:

$$P_h(z|x, m) \propto corr(r(z, x), m) \tag{12}$$

where z is the lazer scan's likelihood and $r(z, x)$ is the scan's projection to the world frame and m is the grid map. The function itself measures the similarity of the scan and the previous time step's map by counting the number of equal predicted occupied cells within a 9x9 cell neighborhood with:

$$corr(y, m) = \sum_i \mathbb{1}\{y_i = m_i\} \tag{13}$$

After map correlation is run on each particle, the particle weights of the particles are updated with:

$$\alpha_{t+1|t+1}^{(k)} = \frac{\alpha_{t+1|t}^{(k)} C_*^{(k)}}{\sum_j \alpha_{t+1|t+1}^{(j)} C_*^{(j)}} \tag{14}$$

Where $C_*^{(k)}$ is the max correlation value of all particles and $C_*^{(j)}$ is the j'th particles max correlation value. After the weights are updated, the $i_* argmax(weights)$ is taken and the i_*^{th} particle's pose is updated as the new predicted pose.

Next, based on the new predicted pose (max correlation particle's pose), the log odds map is updated via the 2D Bresenham function, which calculates points that a straight line passes through given start and end points. For points passing through, the cells are deducted by a trust factor (e.g $\log(9)$) indicating the amount of trust placed on the sensors, and for end points, the cells are increased by the trust factor. For preventing over confidence, the log odds map is also clipped at minimum and maximum thresholds. The binary occupancy map is then updated by a threshold on the log odds map.

$$MAP_{Occupancy} = MAP_{LogOdds} > thresh \tag{15}$$

For this implementation the threshold is set to 0. To prevent particle depletion, which is defined as a situation in which most of the updated particle weights become close to zero because the finite number of particles is not enough, re-sampling is implemented. It is applied when the effective number of particles is less than a threshold:

$$N_{eff} := \frac{1}{\sum_{k=1}^{N} (\alpha_{(t|t)}^{(k)})^2} \leq N_{threshold} \tag{16}$$

Where all particle positions are set to the max weight particle's pose and all new weights are set to $\frac{1}{N}$.

E. Technical Approach: Texture Mapping

For texture mapping, the stereo camera data and parameters are used. Since there are only ≈ 1000, a stereo camera reading is read at most once per 100 samples of encoder data and lidar data reading. For a reading at time t, two pictures are provided and the left is taken as the origin. A function calculating the disparity of the left and right stereo camera is provided and the resulting picture is a single 560×1280 disparity image. Using this disparity image, Z can be calculated:

$$z = \frac{f s_u b}{u_L - u_R} \tag{17}$$

Where disparity $= u_L - u_R$, s_u is an intrinsic camera parameter, f is the focal length, and b is the baseline. With this equation y and x are computed as follows:

$$y = z(\frac{v_L - c_v}{f s_v}) \tag{18}$$

$$x = z(\frac{u_L - c_u}{f s_u}) \tag{19}$$

Where u_L and v_L are left camera pixels, with $(0, 0)$ respectively being the most top left pixel. This is implemented in a vectorized manner by using Numpy's np.meshgrid function. The X, Y, Z coordinates are then transformed into the vehicle frame via (7) defined by given parameters and then transformed into the world frame via (7) again based on the max particle's pose. The RGB values can be retrieved from the left camera image and the map can be indexed to have the same texture.

This whole procedure is vectorized with matrix operations and is not that costly to the overall time, taking roughly ≈ 2 times the time. As such, there is the option to not run this part of the code.

F. Technical Approach: Optimization and Conclusion

To optimize the run-time of the code, for-loops were avoided whenever possible, vectorization was employed whenever possible using Numpy's arrays data structure, and data was preprocessed whenever reasonable.

For cleanliness and structure, code was organized into classes such as slam, map, stereo, lidar, helpers, utils. etc. and the main script imports everything to be run and commented to indicate what's happening. Easy parameter configurations of the map size, resolutions and other initializations were also made modularized due to this.

IV. Results

A. Analysis and Images

The overall results are very positive, and increase in accuracy with respect to the amount of Gaussian noise added the particles, the number of particles, and the sampling rate. It was found that the implementation was quite sensitive to noise when the number of particles was small. Using a diagonal co-variance matrix with values anything more than $[5e-5, 5e-5, 5e-5]$, yielded weird paths. It was also found that prediction with the differential motion model itself yielded a reasonable path without even considering motion model and localization and mapping. However, this does not provide a map of occupied and free cells. When adding more particles, the results are even better albeit slower since the time complexity is linear with respect to particles.

Below are some of the results including (a) images of the trajectory and occupancy grid map over time and (b) textured maps over time.

B. Conclusion

Overall, the implementation of particle filter SLAM and texture mapping was a success. For future improvements, asynchronicity of the sampling can be implemented in a fashion similar to ROS, where instead of a for-loop looping over the data indices, each sampling time is read and the next reading in chronological order is used. This can improve the syncing of the data and perhaps the results of the prediction. Secondly, re-sampling can be done in a stratified manner, instead of assigning all particles to the max, they can be assigned to locations with respect to the weights.

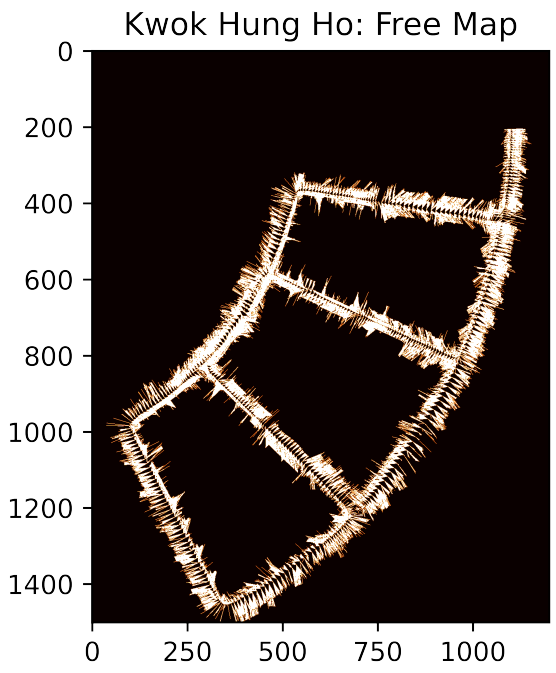

Fig. 1. Free Space Map

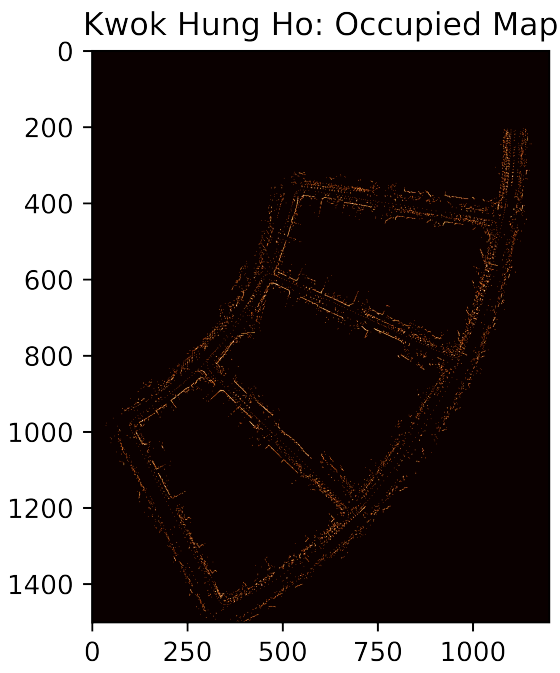

Fig. 2. Occupied Space Map

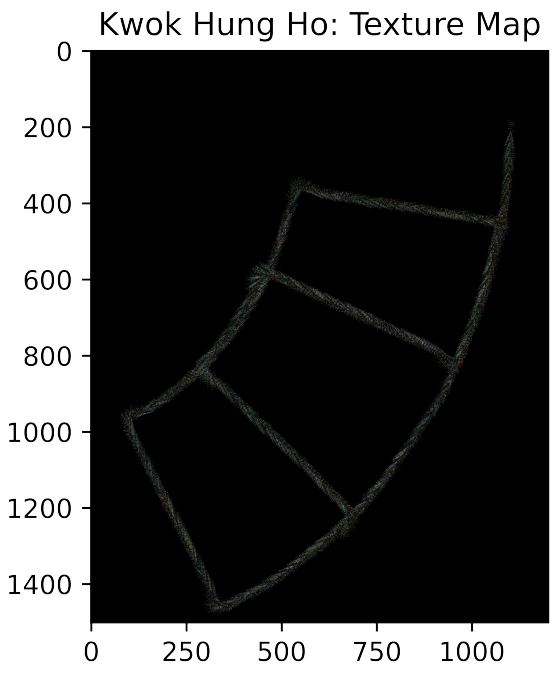

Fig. 3. RGB Texture Map

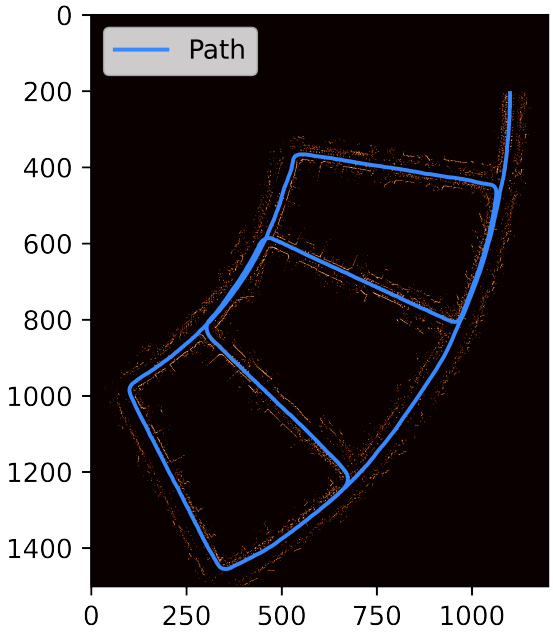

Fig. 4. Occupied Map with Path 1

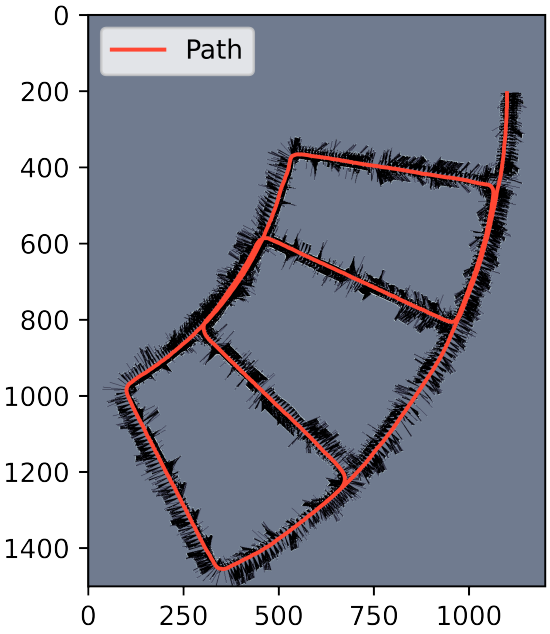

Fig. 5. Occupied Map with Path 2

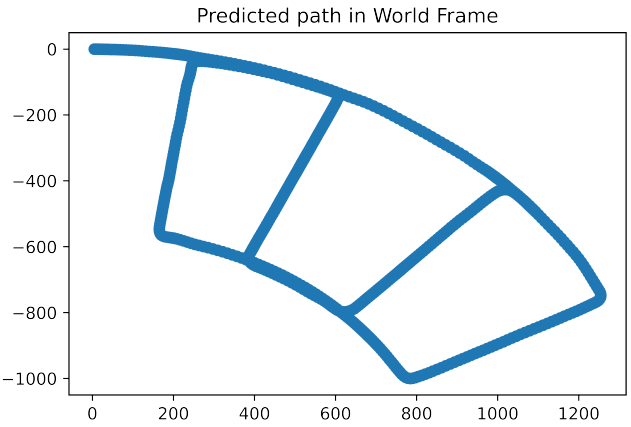

Fig. 7. Predicted Path in World Frame

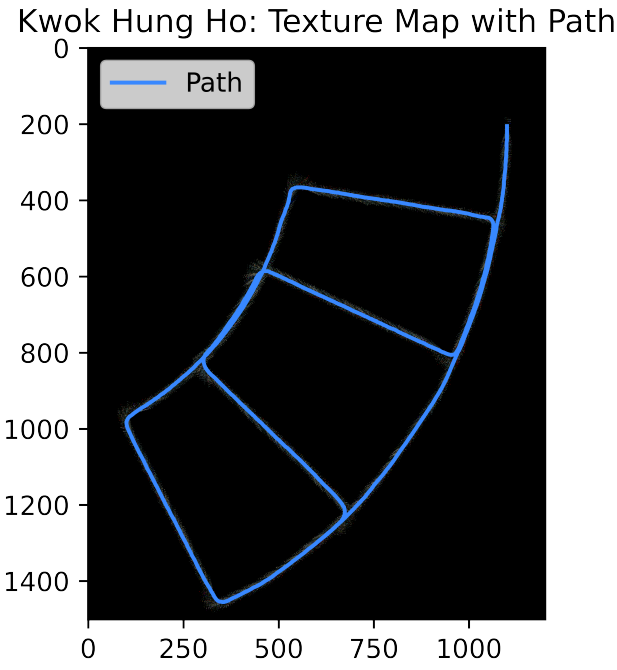

Fig. 6. Texture Map with Path

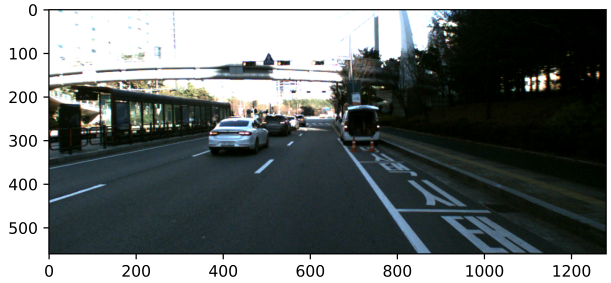

Fig. 8. Example of Stereo Image (right camera)