Particle Filter SLAM for creating the MAP

Harsh Kishorsingh Thakur
Department of Electrical and Computer Engineering
University of California, San Diego
La Jolla, U.S.A.
HThakur@ucsd.edu

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

Abstract— The main idea behind this project is sensing part of the Robot. In order to do such sensing, I am assuming that the robot has a camera embedded in it and it takes pictures of the surrounding. Now the goal of this paper is to classify the image find a blue color and check whether this blue color represents the recycling bin or not. In order to fulfill this goal, it is necessary to design an unconstraint optimization algorithm that will help us solve this problem.

I. Introduction

Nowadays, robots have become an integral part of society. One of the most common robots that you can see around and everyone is familiar with it is the robot vacuum cleaner. So, you need a map of the world in order to vacuum at all the spots of the world. But, we don't how each individual home looks like or where they have places stuff.

This is where **Simultaneous Localization And Mapping (SLAM)** comes into the picture. The basic idea of SLAM is to predict the next location of the robot which is accurate enough to resemble the current world. There are multiple ways in which this can be done those are Rao-Blackwellized Particle Filter, Kalman Filter, Factor Graphs SLAM, Fast SLAM[1], Kinect Fusion. All this method helps us to build a map of the environment. Now, these maps can either be Sparse or Dense depending on our requirement. Point cloud maps, landmark-based, and surfels are some examples of sparse maps. While on the other hand, there are two map types in the dense map, i.e. Implicit surface model, e.g. Occupany map and distance-based map and explicit surface models, i.e. polygon mesh.

Now my goal is to make an autonomous car that has Stereo Camera, 2D LiDAR and 3D LiDAR sensor, IMU, and Encoder. So, IMU and Encoder will help me to get the speed of the car, or theoretically will help me find the next position of the car before even going there. And the camera and 2D LiDAR sensor will help me get the map of the environment. Thus, where ever the car is going I will create the map of the free space and occupied space on the vehicle trajectory.

II. Problem Formulation

2.1 Autonomous Vehicle Configuration:

The autonomous vehicle given to us has a certain configuration. In order to map the environment, we want the position of the car in the real world.

Here the sensor reading is according to their perspective or they are in the sensor frame and we need them from world perfective thus there are different transformation that needs to be done in order for the world to know what sensors means to the world. Each sensor are are at a different location which can be seen from the image below.

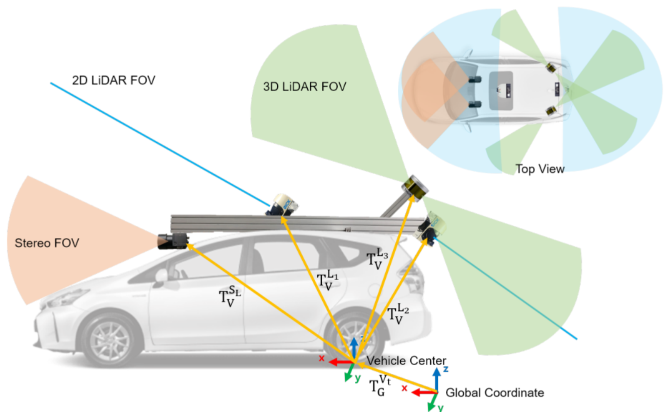


Fig 1. Sensor Position

Considering the center of the car at the rear wheel so defining the position of the other sensors accordingly. The IMU and encoder which are not shown in Fig 1. are shown in Fig 2. The IMU is located 4.98m in the front, from the center of the car. And it gives us the value $[\Delta\text{roll}, \Delta\text{pitch}, \Delta\text{yaw}]^T \in R^3$, for our case as the car can be considered as moving on a flat surface so we will neglect the values of the roll and pitch and just take into consideration the yaw, which in turn help me to find the orientation of the car in the real world.

The encoders are present on the left and right rear wheel and the data in the format of [left, right] $\in R^2$, this gives me the value of the left wheel and right wheel encoder respectively. Using this I find the speed of my vehicle.

The 2D LiDAR Scanner that we are going to use is at a height of 1.82m and 0.758m in the front, from the vehicle center. The Field of view of the LiDAR sensor is 190° i.e. -5° to 185°, with an angular resolution of 0.666° and the max range the LiDAR can detect an object is 80m.

The stereo camera is located in the front of the vehicle at a height of 1.77m from the center of the car. And at each time stamp we get images from a left and a right stereo camera, thus combining them together we get the RGB image as well depth in an image.

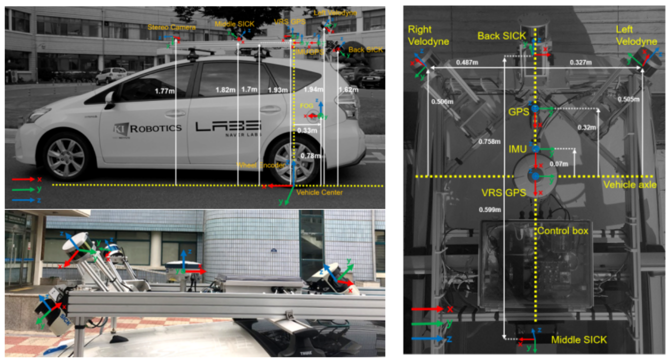


Fig 2. Sensor Position

Also, all the sensors are not in a synchronous manner, but I am given the timestamp of each data, thus like timing the data is required in order for the sensor to work altogether.

2.2 Simultaneous Localization And Mapping (SLAM)

There is always some kind of some in the environment. For example, noise in the encoder or in general inaccurate measurements given by sensors. Thus the whole trajectory becomes probabilistic. And this is defined by the probabilistic Makov assumption.

Thus SLAM comes into the picture. Let the robot position, control unit, and noise at time t be x_t, $u_{t,}$ and w_t respectively. The next predicted step becomes:

$\mathbf{x_{t+1} = f(x_t, u_t, w_t) \sim pf(\cdot|x_t, u_t)}$, and this is basically called the motion of the robot. It can either be written as a linear function or can be written as a probabilistic model.

Now once we know the next predicted step of our vehicle, we can observe the environment with the camera and the LiDAR sensor data. And, eventually, try to correlate with the previous map that I have created. Let the observation and noise at time t be z_t and v_t respectively. Thus our observation model is

$\mathbf{z_t = h(x_t, m, v_t) \sim ph(\cdot|x_t, m)}$, where m is the map until time t.

2.3 Bayesian Inference and Filtering

Now, we know the motion model and the observation model how do we calculate x_{t+1} and z_t. This is where Bayes Filter comes in place.

There are basically two steps involved in this:

1. Given robot position and control input what will be the next position? **Prediction**
2. Given the position and previous observation, what is the new observation? **Update**

Prediction:

For prediction given the prior pdf or pmf, in our case although it is kind of continuous we take a small difference in time and try to make it discrete. Thus given a pmf at time t i.e. $p_{t|t}$ we try to compute $p_{t+1|t.}$ Thus we get the equation

$\mathbf{p_{t+1|t}(x_{t+1}) = \int pf\,(x_t|\ s,\ u_t)\ p_{t|t}\,(s)ds.}$

Update:

For update state, we get a new observation, and thus in that case using the updated position of my vehicle in predict step, and the new observation we try to correct our position to correct location.

$$\mathbf{p_{t+1|t+1}(x_{t+1}) = \frac{ph(z_{t+1}|x_{t+1})\ p_{t+1|t}(x_{t+1})}{\int ph(z_{t+1}|s)\ p_{t+1|t}(x_t)\ ds}}$$

2.4 Occupancy Grid Map

Occupancy grid map helps us to find the places where objects are located. Thus, when my vehicle is moving on the road, using the Lidar Data at time t, I update the map. Thus occupancy map can be said as like a big 2D map that has nxn cell. Each cell represents a pixel. Now if there are objects in that cell we make it distant from the free or unobserved location.

Let's supposed that a particular cell is free and I represent it as m = +1 if occupied and m = -1 if unoccupied and the equation is:

$\mathbf{p(m\ |\ z_{0:t}\ ,\ x_{0:t}\ ),}$ Here $Z_{0:t}$ represents the observation up to time t and $X_{0:t}$ represents the position up to time t.

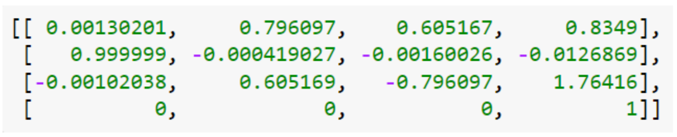
One assumption we make for our model is that all the cells are independent of each, that is if the cells next to each can be unoccupied or occupied only if we observe it. This is not true in general because as in the real world a particular object takes many cells and corresponding cells can be stated as occupied.

III. Technical Approach

3.1 Initializing the map (Mapping):

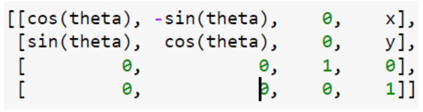
In this problem, I don't know the map or the starting point of the car in the world frame I am free to initialize from the position (0, 0, 0) i.e. (x, y, theta), X and Y are the coordinates of the map and theta is the angle at which the car is moving.

In order to initialize the map, I used the LiDAR data used the occupancy map part of it. But the LiDAR data I have is in the LiDAR frame so, first, we need to convert it into the vehicle frame, this is done using the transformation matrix given to us. Thus we use the pose of the lidar to vehicle frame as. ${}_{V}T_{L}$ =



```
[[ 0.00130201,      0.796097,     0.605167,      0.8349],
 [   0.999999, -0.000419027, -0.00160026, -0.0126869],
 [-0.00102038,      0.605169,    -0.796097,     1.76416],
 [          0,             0,            0,           1]]
```

Also we need to convert the vehicle frame to the world frame we do this by using the general form ${}_{W}T_{V}$ =



```
[[cos(theta), -sin(theta),    0,    x],
 [sin(theta),  cos(theta),    0,    y],
 [         0,           0,    1,    0],
 [         0,           0,    0,    1]]
```

Where x, y, and theta are from the particle. Thus this matrix changes every iteration. And we have lidar data, but we just know the distance from the lidar data to the endpoint, so in that, we use geometry to convert the distance into a matrix form. Thus the x coordinate of all lidar data is distance*cos(angle) at that particular and the y coordinates are distance*sin(angle). Once we get all the endpoint of the lidar data in the lidar frame we do calculations like this.

$S_W = {}_{W}T_{V} * {}_{V}T_{L} * S_L$.

Now using the occupancy grid technique we try to create the map. First of all, I just have the endpoint of the lidar data. I am provided with a given function that is bresenham2D, which finds all the coordinates between the starting point and endpoint. Using this function, I add log(4) and for the rest of the coordinates between the lidar starting point and lidar ending point, I subtract log(4). This is a trick to avoid integration.

As we don't know anything about the map, we initially start the map with all zero, thus the given prior is unknown.

$$p(\mathbf{m} \mid \mathbf{z}_{0:t}, \mathbf{x}_{0:t}) = \prod_{i=1}^{n} p(m_i \mid \mathbf{z}_{0:t}, \mathbf{x}_{0:t})$$

. Given all the calculations in the class, we came to a conclusion that, If we think our sensors are 80% correct, we can consider

$$\Delta\lambda_{i,t} = \log \frac{p(m_i = 1 \mid \mathbf{z}_t, \mathbf{x}_t)}{p(m_i = -1 \mid \mathbf{z}_t, \mathbf{x}_t)} = \begin{cases} +\log 4 & \text{if } \mathbf{z}_t \text{ indicates } m_i \text{ is occupied} \\ -\log 4 & \text{if } \mathbf{z}_t \text{ indicates } m_i \text{ is free} \end{cases}$$

Thus by doing this trick we find the occupancy map and initialize the world. This becomes now the reference map, and we know where our vehicle is in the world frame.

3.2 Prediction Step:

In the prediction step, using the sensor data of encoder and imu we find the next coordinate of our car. So initially we started from (0, 0, 0) and we try to update the position of the car. So, I know that my car is following a differential drive model. Therefore our linear function $\mathbf{x_{t+1} = f(x_t, u_t, w_t) \sim pf(\cdot | x_t, u_t)}$ is a differential drive function, but as the noise here is a random variable it becomes a probabilistic function.

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

In this equation, x_t is the position of the car at the previous time. V_t and w_t is the velocity and angular velocity given at time t respectively. Here the angular velocity directly comes from FOG data that is provided to me.

But, in order to find the velocity of the car, we have to use the model for the encoder. I am given the encoder reading so combining with how the encoder works, in a given time frame: velocity of a particular wheel becomes

v = (π*d*Δz) / 360 *Δt. Thus doing this for the left wheel and right wheel and taking the average we get the linear velocity. Now we add Gaussian noise of zero mean, to the car so that there is some randomness and assume the sensor to be not 100% correct. Thus in the prediction step, we get the new position of the car.

3.3 Update Step:

Now from the prediction step I know the next position of the car, once I know that I create a temporary map using the new lidar data point at that timestamp, I do the same steps as described in 3.1 Initializing the map instead of the initial point (0, 0, 0) I will have a different position, I apply all the transformation on the LiDAR data and create a temporary map. After that, I check whether the current map is having a correlation with the original map or not. This doesn't make sense if I have just one particle, but if i have 100 particles and they all have different positions, I find the best particle out of this 100 using the correlation and thus I update the map.

This idea is extended from the update step of 2.3 Bayesian Filter i.e. $\mathbf{p_{t+1|t+1}(x_{t+1})}$. The pmf we calculate is again a mixture of the delta functions. And the important part comes from the correlation function.

$$
\begin{aligned}
p_{t+1|t+1}(\mathbf{x}) &= \frac{p_h\left(\mathbf{z}_{t+1} \mid \mathbf{x}\right) \sum_{k=1}^{N_{t+1|t}} \alpha_{t+1|t}^{(k)} \delta\left(\mathbf{x}-\boldsymbol{\mu}_{t+1|t}^{(k)}\right)}{\int p_h\left(\mathbf{z}_{t+1} \mid \mathbf{s}\right) \sum_{j=1}^{N_{t+1|t}} \alpha_{t+1|t}^{(j)} \delta\left(\mathbf{s}-\boldsymbol{\mu}_{t+1|t}^{(j)}\right) d\mathbf{s}} \\
&= \sum_{k=1}^{N_{t+1|t}} \underbrace{\left[\frac{\alpha_{t+1|t}^{(k)} p_h\left(\mathbf{z}_{t+1} \mid \boldsymbol{\mu}_{t+1|t}^{(k)}\right)}{\sum_{j=1}^{N_{t+1|t}} \alpha_{t+1|t}^{(j)} p_h\left(\mathbf{z}_{t+1} \mid \boldsymbol{\mu}_{t+1|t}^{(j)}\right)}\right]}_{\alpha_{t+1|t+1}^{(k)}} \delta(\mathbf{x}-\underbrace{\boldsymbol{\mu}_{t+1|t}^{(k)}}_{\boldsymbol{\mu}_{t+1|t+1}^{(k)}})
\end{aligned}
$$

Now, suppose instead of 1 particle we have N particles. In this case, initially, we start with equal weights. Each particle has a weight of 1/N. So, using correlation we find the best particle. I pass the temporary map to the map-correlation function and it returns me a matrix. Now, if I want a range of matrices, I can play with the parameters, but as I am just picking the center position I update my weight as weight=weight*exp(center_position_value). Once I get the matrix, I store all the center points of this correlation matrix. Once I iterate over each particle, I pick the best correlation value. And I consider as my best particle. And finally, update the original map according to this particle position. Once before leaving the update step, I renormalize the weights, so they add up to 1.

3.4 Resampling step:

Due to noise term in the motion model, a few particles might deviate a lot and thus their weight becomes negligible and we don't consider them, thus when something like that happens. I try to resample the particles and bring them back to the particles that are having higher weight.

I put a threshold on a total number of particles, now I check how many particles are effective at time t. And it can be calculated using this formula. After the update step we know what new alpha is i.e. weights are, I can check how many particles are effective. If they are less than the threshold, I distribute the weights equally and bring extremely deviated particles to particles with higher weights.

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

3.5 Texture Map:

The main part of txture mapping is getting the image from the stereo camera and get the depth of the position. This is done by the function provided to me i.e. stereo_camera.

Once I have the disparity image. I can easily find the Z_0 from the fs_u and the baseline between the camera. Find X_0 and Y_0 from the formula of disparity measurement. After getting the disparity I can find the caressponding X and Y axis using the Stereo Camera model. Once I get the coordinates X, Y and Z in optical frame. I use the transformation

$$d = u_L - u_R = \frac{1}{z} fs_u b$$

$[x_w, y_w, z_w, 1] = {}_wT_r * {}_rT_o * [X_o, Y_o, Z_o, 1]$.

After transforming to world frame. We have to find the texture which is close to the road. The Z_w provides us with the height in the world frame, I just take the pixel that are close to ground. Once I have the coordinates I take the region of interest. And transform back to the optical frame. Using the u, v values in the optical fram, I use the left stereo image to get the RGB data at that location. And update the texture map.

3.6 Pseudo-Code of SLAM:

The pseudo-code of my program looks like this.

```
SLAM
01: Initialize the total number of particles, assign them equal weights, and
    all particles start from (0, 0, 0)
02: Initialize the empty occupancy map and log map.
03: Using the first lidar data, transform from lidar to world frame, get
    empty space using the bresenham2D function. And merge this data
    with the empty map.
04: for i in range(all_data):
05:        predict the new position of these particles using the motion
    model.
07:        I am updating after 5 predictions.
08:        update step()
09:        In the update step, I take new lidar that resembles the to time t.
10:        Transform this lidar data to the world frame and do all the
    transformation and everything and update the wights using the map
    correlation.
11:        As the weights are only updated after 5 iterations, resampling is
    also done after that.
12:        Resampling step
13:        And after every 100 iterations I take a photo so at the end I can
    make a video, which is included in a zip file.
14:
15: for i in range(stereo_image):
16:        Match the closest Lidar Data to it.
17:        As I have stored the whole robot trajectory in an array, I find the
    position at this timestamp.
18:        Get the Z = (fs_u * baseline) / disparity image.
19:        Transform the image pixel from optical to world frame using the
    transformation matrices.
20:        Get the points that are close to the ground using the Z axis.
21:        Once I get the desired points, I transform those points into the
    optical frame again. Just so, that I can get the RGB values from those
    coordinates.
22:        And at the end I add those to my texture map.
```

IV. Evaluation

I have the data from the encoder and IMU using that I can get the approximate path of the model. And in update step, I check the correctness of the path using the lidar data.

Npw in order to check how accurately my model is working for the given data, I test it for two different noise and different number of particles.

4.1 Dead Reckoning

This is what my dead reckoning looks like without any noise.

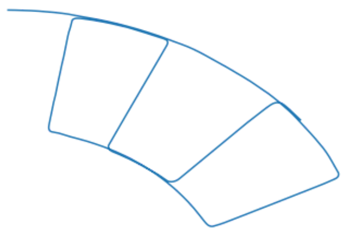


Fig3. Dead Reckoning

4.2 First Lidar Data

This is how my first LiDAR scan looks like.

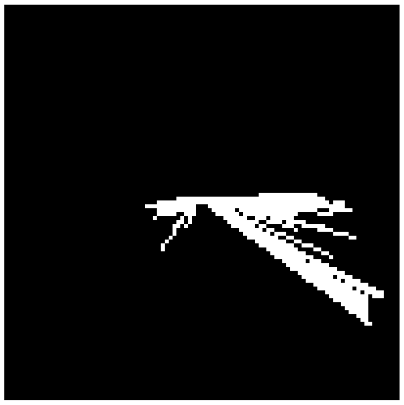


Fig 4. First Lidar Data

4.1 Number of Particle = 10:

Here The noise was sufficient enough so when a single particle is travelling with that noise it deviated from the values received from the encoder and data.

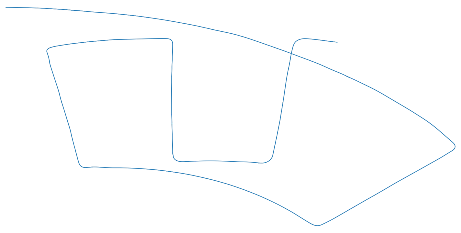


Fig 5. Noise Map

The hyper-parameters are as follows:-

Number of Particles	10
log_min	-1.5
log_max	1.5
Noise in velocity	1e-10 m/nanosecond
Noise in omega	1e-12 rad/nanosecond
Lidar Rays Max	60
Lidar Rays Min	0.1
Map Resolution	1
X_MIN	-300
X_MAX	1500
Y_MIN	-1500
Y_MAX	400

Table 1. Parameters for Test 1

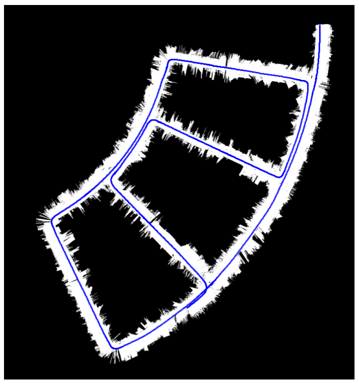


Fig 6. Free Map

I have also put the video of this in the zip file name particle "**10_particle.mp4**".

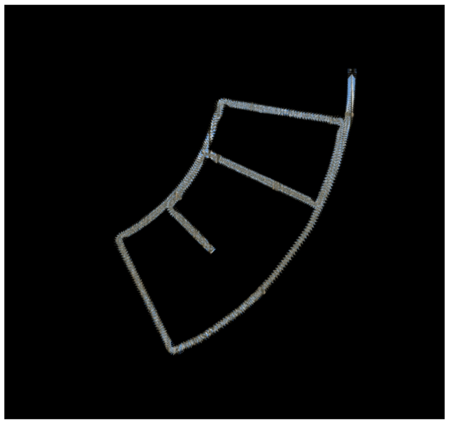


Fig 7. Texture map

4.2 Number of Particle = 100:

In this the noise was extreme even though the LiDAR data helped in choosing amost the right correct trajectory of vehicle.

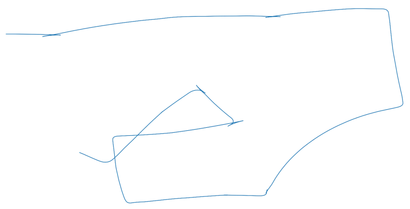


Fig 5. Noise Map

The hyper-parameters are as follows:-

Number of Particles	100
log_min	-3
log_max	3
Noise in velocity	5e-10 m/nanosecond
Noise in omega	5e-12 rad/nanosecond
Lidar Rays Max	60
Lidar Rays Min	0.1
Map Resolution	1
X_MIN	-300
X_MAX	1500
Y_MIN	-1500
Y_MAX	400

Table 2. Parameters for Test 2

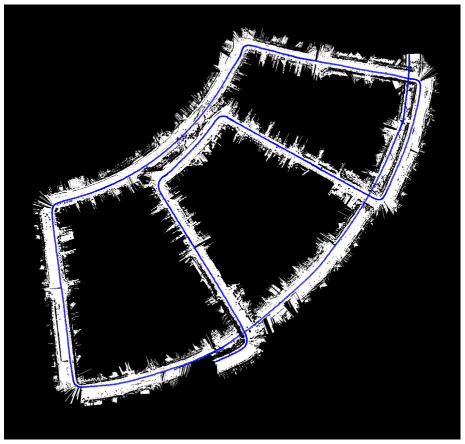


Fig 8. Free Map

I have added the video of this in zip file names "**100_particles.mp4**".

One Important thing I found while making log map, was if I dont threshold the log values, when the car stop at a particular traffic light. The log values got increased a log and hence when it tries to move to next position, the correlation map value became zero. Because it could not match small values with the large values, thus clipping of log values is required.

Another important thing I noticed is adding noise is okay, but when the car is at stop adding noise doesn't make sense that much because, when the car is at stop the encoder reading can't be wrong. But even then I added the noise in order to reduce the number of lines in the codes.

References

[1] http://robots.stanford.edu/papers/montemerlo.fastslam-tr.pdf
[2] Discussed with Sambharan.
[3] A lot of equations are from class slides.
[4] Video of my free map at 10_particles.mp4
[5] Video of my free map at 100_particles.mp4