SLAM System and Texture Mapping

Tongji Luo
Department of Electrical Computer Engineering
University of California, San Diego
ltongji@ucsd.edu

***Abstract*—This paper presented approaches using SLAM system to solve the localization problem of a differential-drive robot and using depth and RGB camera to texture the map. This work shows the detailed steps of using particle filter method to localize the accurate location of the robot and bulit the map detected by the robot. Finally texture the map with real color segment.**

***Index Terms*—Particle filter, RGBD measurements, Texture mapping.**

I. Introduction

The problem of simultaneous localization and mapping (SLAM) of mobile robots is a key problem that mobile robots must solve in realizing autonomy in an unknown environment. In recent decades, with the continuous development of robotics and artificial intelligence technology, and the increasing demand for service robots in the whole society, academics and industry have invested a lot of resources to conduct in-depth research and application exploration of robot navigation technology. SLAM becomes one of the research focus and hotspots of mobile robotics.

Simultaneous localization and mapping (SLAM) requires the robot to start from an unknown location in a completely unknown environment, and use the sensor to observe the environment to incrementally establish the navigation map of the environment. At the same time, according to the established map synchronization to determine their position, and thus comprehensively answer the question "Where am I?" and finally evaluate the surrounding environment.

In this problem, we are provided with folllowing data: encoder counts of the robot wheels, Yaw velocity from IMU sensor, lidar scan of the environment and the RGBD measurements. Our goal is to solve the SLAM and texture mapping problem using the above data and acquire the map of the environment. Our approach can be divided into following four steps:

- Trajectories of the robot
- Mapping
- Particle filter
- Texture mapping

A. Trajectories

In this part, we arrange the encoder stamps, IMU stamps into the same time line. We use the time as trigger to record the encoder changes(distance of the robot movement during this time) and find the avarge IMU changes between neighboring encoder stamps to get the yaw of the robot. Finally use the differential motion model to compute the trajectories of the robot.

B. Mapping

In the mapping part, we put the lidar scan data in the same time line with the robot trajectories. After transforming the lidar ranges data and the robot position to the map frame, we can acquire the log-odds of each point in the map we set up. From constantly upadating, we can have the map of the obstacle scanned by the lidar.

C. Particle filter

Set up N particles from the original point (0, 0, 0), using the same method of tracking robot trajectories and mapping to update the position of each particles and find the best one as the accurate position of the current robot and update the log-odds map.

D. Texture Mapping

Using depth camera as media between the ground world and the RGB camera to match the RGB index of the pictures to the world position. Draw the color on the specific area of the map to finish the texturing.

II. Problem Formulation

A. Transformation of the frame

First we should transform the lidar scanned data to the body frame, notice that the lidar and the body center have the same y axis, so we set up the transformation matrix as:

$$\begin{bmatrix} 1 & 0 & 0 & 133.23e^{-3} \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 514.35e{-}3 \\ 0 & 0 & 1 & 1 \end{bmatrix}$$

to transform the lidar frame to the body frame.

The second transformation is to translate the body frame to the world frame, we take the current location of the robot (x, y, θ) and the transformation matrix is as:

$$\begin{bmatrix} cos(\theta) & -sin(\theta) & 0 & x \\ sin(\theta) & cos(\theta) & 0 & y \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix}$$

B. Trajectories

To get the trajectories of the robot, we need to compute the distance and orientation of robot movement at each time step. Take a encoder counts at one time as: [FR, FL, RR, RL]. The velocity of the robot can be computed as:

$$\text{v = (FR + FL + RR + RL)/4*0.0022}$$

The discrete-time differential-drive model can be represented as following:

$$\begin{aligned}
\theta(t) &= \theta(t_0) + \int_{t_0}^{t} \omega ds = \theta(t_0) + \omega(t - t_0) \\
x(t) &= x(t_0) + v \int_{t_0}^{t} \cos\theta(s) ds \\
&= x(t_0) + \frac{v}{\omega}\left(\sin\left(\omega(t - t_0) + \theta(t_0)\right) - \sin\theta(t_0)\right) \\
&= x(t_0) + v(t - t_0)\frac{\sin(\omega(t - t_0)/2)}{\omega(t - t_0)/2}\cos(\theta(t_0) + \omega(t - t_0)/2) \\
y(t) &= y(t_0) + v \int_{t_0}^{t} \sin\theta(s) ds \\
&= y(t_0) - \frac{v}{\omega}\left(\cos\theta(t_0) - \cos\left(\omega(t - t_0) + \theta(t_0)\right)\right) \\
&= y(t_0) + v(t - t_0)\frac{\sin(\omega(t - t_0)/2)}{\omega(t - t_0)/2}\sin(\theta(t_0) + \omega(t - t_0)/2)
\end{aligned}$$

C. Particle filter

First we use the mixture of delta functions with weight $\alpha^{(k)}$ to represnt the pdfs of the particles:

$$\delta(x; \mu^{(k)}) := \begin{cases} 1 & x = \mu^{(k)} \\ 0 & \text{else} \end{cases} \qquad \text{for } k = 1, \ldots, N$$

The prior distribution of the particles is:

$$x_t \mid z_{0:t}, u_{0:t-1} \sim p_{t|t}(x_t) := \sum_{k=1}^{N_{t|t}} \alpha_{t|t}^{(k)} \delta\left(x_t; \mu_{t|t}^{(k)}\right)$$

The prediction of the probability of the particles is:

$$p_{t+1|t}(x) = \sum_{k=1}^{N_{t|t}} \alpha_{t|t}^{(k)} p_f(x \mid \mu_{t|t}^{(k)}, u_t) \approx \sum_{k=1}^{N_{t+1|t}} \alpha_{t+1|t}^{(k)} \delta\left(x; \mu_{t+1|t}^{(k)}\right)$$

Then we do the update step the adjust the weight of each particles based on the observation likelihood:

$$p_{t+1|t+1}(x) = \sum_{k=1}^{N_{t+1|t}} \left[\frac{\alpha_{t+1|t}^{(k)} p_h\left(z_{t+1} \mid \mu_{t+1|t}^{(k)}\right)}{\sum_{j=1}^{N_{t+1|t}} \alpha_{t+1|t}^{(j)} p_h\left(z_{t+1} \mid \mu_{t+1|t}^{(j)}\right)}\right] \delta\left(x; \mu_{t+1|t}^{(k)}\right)$$

For this observation likelihood, we use **mapCorrelation** to get the correlation of each particles with the map. Then we can update the weight using:

$$p_h(z_{t+1} \mid \mu_{t+1|t}^{(k)}, m) \propto \exp\left(\text{corr}\left(y_{t+1}^{(k)}, m\right)\right)$$

After each update step, we should compute the nubmer of the efficient particles and see if that number is smaller than the threshold number we set up:

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

Which means the rest particles can not effctively represent the true trajectories of the robot, we will do the **resample** to use the particles which have high weights to replace that have lowest weights.

D. Mapping

Occupancy grid map is a grid m with free($m_i = 0$) and occupied($m_i = 1$) cells. When we are updating the cells'(m_i) pdfs, is equivalent to accumulating the log-odds ratio:

$$\begin{aligned}
\lambda(m_i \mid z_{0:t}, x_{0:t}) &:= \log o(m_i \mid z_{0:t}, x_{0:t}) = \log\left(g_h(z_t \mid m_i, x_t) o(m_i \mid z_{0:t-1}, x_{0:t-1})\right) \\
&= \lambda(m_i \mid z_{0:t-1}, x_{0:t-1}) + \log g_h(z_t \mid m_i, x_t) \\
&= \lambda(m_i) + \sum_{s=0}^{t} \log g_h(z_s \mid m_i, x_s)
\end{aligned}$$

We can updat the cells based on how much we trust the occupancy measurement(lidar scanned results and bresenham2D function):

$$g_h(1 \mid m_i, x_t) = \frac{p_h(z_t = 1 \mid m_i = 1, x_t)}{p_h(z_t = 1 \mid m_i = 0, x_t)} = \frac{80\%}{20\%} = 4 \qquad g_h(0 \mid m_i, x_t) = \frac{1}{4}$$

Thus, we add occupied cells with log(4), the free cells with $\log(\frac{1}{4})$.

E. RGBD measurement

For the texture mapping part, first we obtain the **d** value of each pixel from the disparity image. Then we can transform it into the optical frame by:

$$\underbrace{\begin{pmatrix} u \\ v \\ 1 \end{pmatrix}}_{\text{pixels}} = \underbrace{\begin{bmatrix} fs_u & fs_\theta & c_u \\ 0 & fs_v & c_v \\ 0 & 0 & 1 \end{bmatrix}}_{\text{calibration: } K} \underbrace{\frac{1}{Z_o}\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix}}_{\text{canonical projection: } \Pi_0} \begin{pmatrix} X_o \\ Y_o \\ Z_o \\ 1 \end{pmatrix}$$

In which, $\frac{1}{Z_0}$ can be considered as depth, obtained by:

$$dd = (-0.00304 * d + 3.31)$$
$$depth = \frac{1.03}{dd}$$

After that, we can obtain the (R, G, B) of the pixel $rgb_img[rgbi][rgbj]$ corresponding to (u, v) by:

$$rgbi = (u*526.37 + dd*(-4.5*1750.46) + 19276.0)/585.051$$
$$rgbj = (v * 526.37 + 16662.0)/585.051$$

And apply it to the map.

III. Technical Approach

In this section we discuss about algorithms and models used in this project. The technical approach can be mianly divided into four parts:

- Trajectories of the robot
- Mapping
- Particle filter
- Texture mapping

A. Trajectories of the robot

To get the trajectories of the robot over time, we would mainly use the encoder counts which represents the movement of the wheel and the IMU angular velocity which represents the yaw velocity of the robot.

Notice that the encoder and the IMU have different time stamps, which means we should first arrange them in a certain sequence to get the proper motion of the robot.

In this case, we set up a timeline from the min(encoder time and IMU time) to the max(encoder time and IMU time). The time **t** moves 0.001 second each time, when the t surpass one encoder time, we consider it as one triggering to this encoder and we can get the corresponding velocity of the robot. Then we calculate the mean of the IMU yaw-angualr-velocity between this encoder time and the next time to get the yaw-angualr-velocity for this time. Later we would add the lidar stamps and lidar scan data on this timeline.

Initially, we set the robot at (0, 0, 0) position. After gain the velocity, the length of the time and the corresponding average angualr velocity during this time. We can apply the differential drive mobile to the robot and find the position of the robot after one movement.

The final trajectories of dataset20 are shown as:

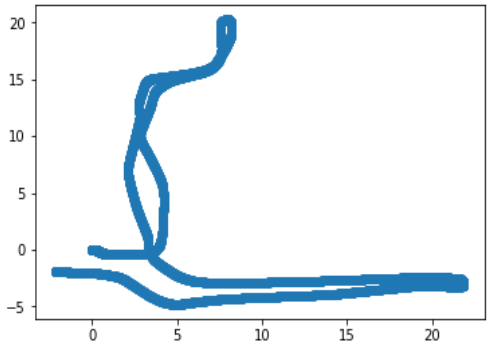

Fig. 1. Trajectories of the robot

B. MAP

Now we add the lidar stamps and lidar scan data on the timeline.

Whenever the lidar be triggered, we take the the current position of the robot (x, y, θ)obtained by the differential drive model and the lidar ranges data as the input.

We first remove the point that are too far or too close, and translate the rest points into the world frame. After transform these points into the MAP cells, we can treat them as the end points of the lidar ray, the start point is the current location of the robot. Then we add log(4) for these occupied cells.

With the above information, we can use the function **bresenham2D** to get the free cells and add log(0.25) for these cells.

This is the whole process for update the map. After that, we would get the new map which has the new odds ratio and apply it to the mapCorrelation part.

C. Particle filter

This part is basically the same with the above parts, just need to add few steps.

This time, instead of setting a single point at (0, 0, 0), we assume there are N(number of the particles) points at (0, 0, 0). Then we set some noise in IMU and encoder data to get N points move to the different spot after one update.

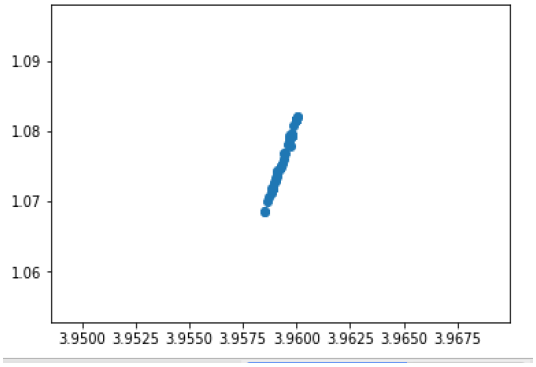

Fig. 2. Particles position after 500 steps

For each of the points, we do a mapping part and get the MAP update by this particle. Then we use the **mapCorrelation** to calculate the correlation rate for this particle to the map. After gaining the N correlation matrix, we can update the weight of each particle for the weight of the particles are proportional to the exponential of the maximum value in their correlation martix.

After gaining the new weight, we use the equation to evaluate if the effctively points are more than the threshold we set up. If not, we do the resample part which is to duplicate the points who have the big weight to replce the ones have really samll weights. The duplicate number are determined by the weight of each particles.

After resampling, we can use the particle who has the biggest correlation value as the best position of the robot to do the mapping part to update the MAP.

D. Texture Mapping

At this step, we also add the disparity and rgb data on the timeline.

We treat the depth camera as a media between rgb camera and the real wold. First, we find the rgb value in the rgb image corresponding to each pixel of the disparity image, and transform the piexl to the boday frame.

Then we use the current robot position (x, y, θ) to transform the rgb coordinates into the world frame, and eliminate the points which have high Z axis values(which means the points are not the ground plane). Applying the (R, G, B) value of each coordinates to the map cells, we can gain the texture map of this step.

IV. Results

A. Single Particle

For the dead-reckoning, we have:

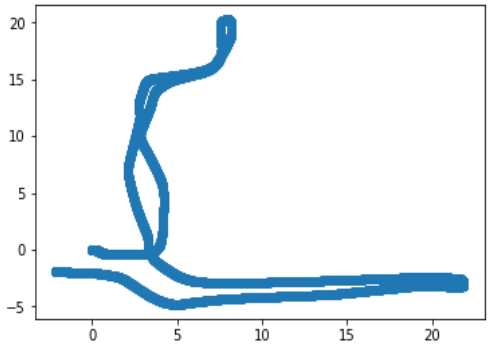

Fig. 3. Trajectories of the robot

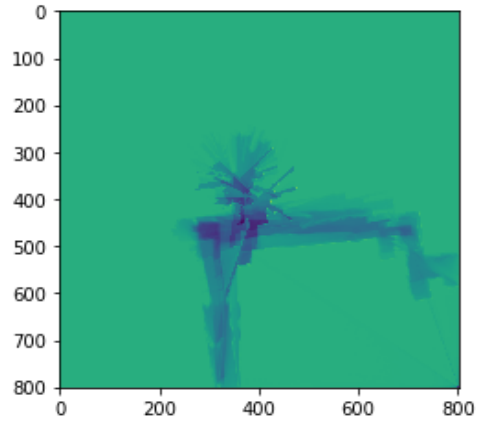

Fig. 4. Map of the single particle

B. Particle filter map over time

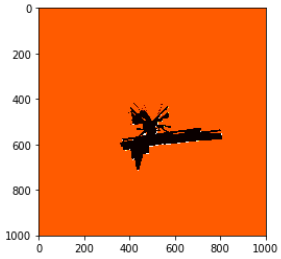

Fig. 6. 500 step map

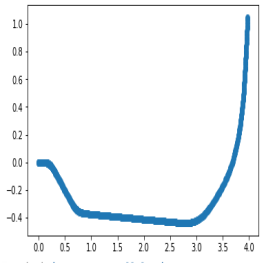

Fig. 7. 500 step trajectories

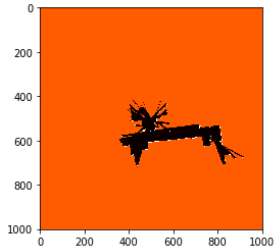

Fig. 8. 1000 step map

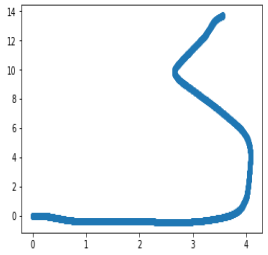

Fig. 9. 1000 step trajectories

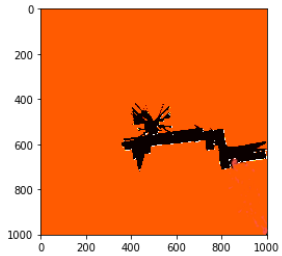

Fig. 10. 1500 step map

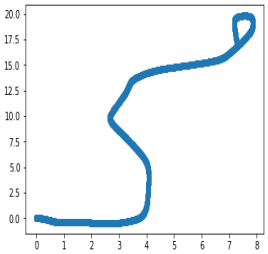

Fig. 11. 1500 step trajectories

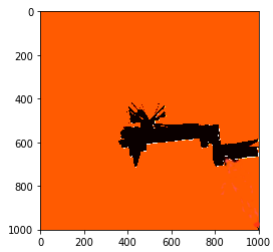

Fig. 12. 2000 step map

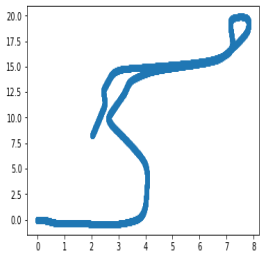

Fig. 13. 2000 step trajectories

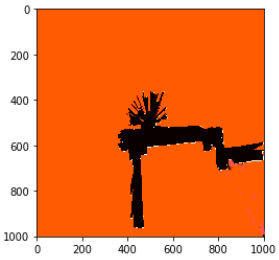

Fig. 14. 2500 step map

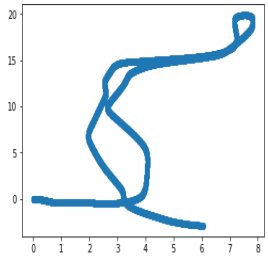

Fig. 15. 2500 step trajectories

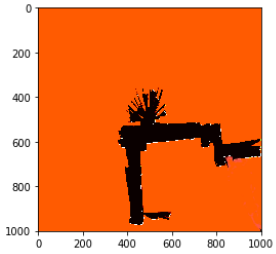

Fig. 16. 3500 step map

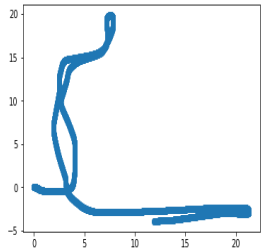

Fig. 17. 3500 step trajectories

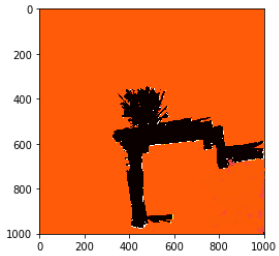

Fig. 18. 4500 step map

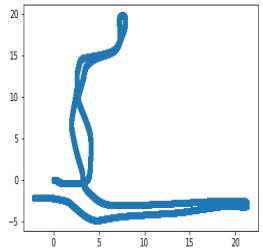

Fig. 19. 4500 step trajectories

C. Results of dataset21

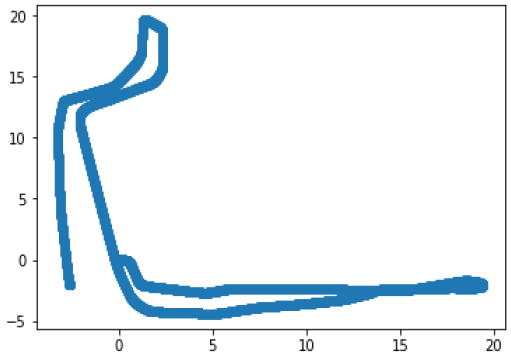

Fig. 19. Trajectories of the robot

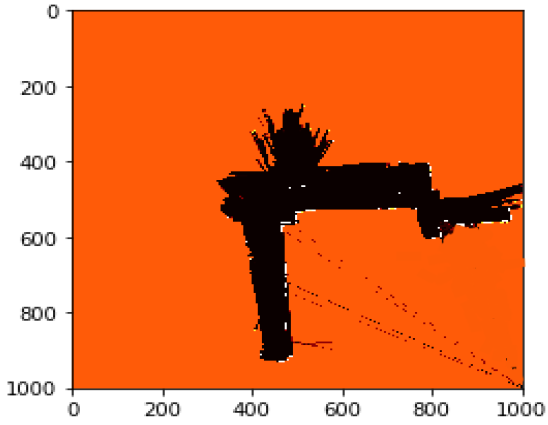

Fig. 20. Map of the environment

D. Results of the testset

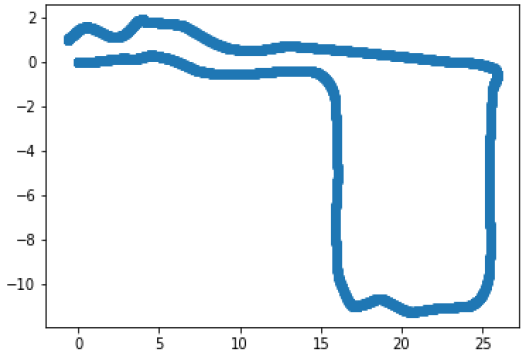

Fig. 21. Trajectories of the robot

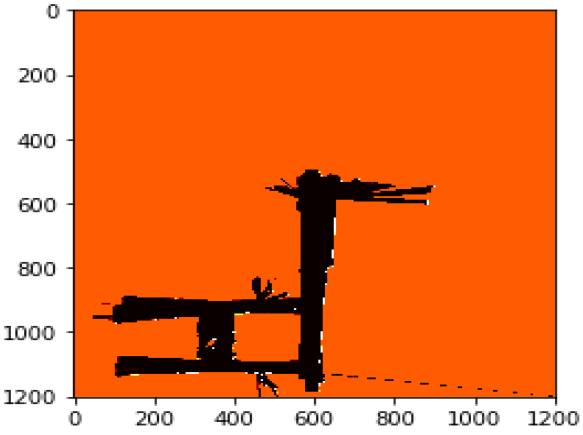

Fig. 22. Map of the environment

E. Result of the texture mapping of dataset20

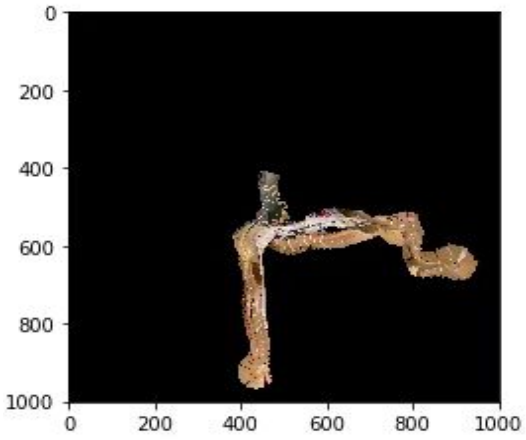

Fig. 23. Texture mapping of dataset 20

F. Result of the texture mapping of dataset21

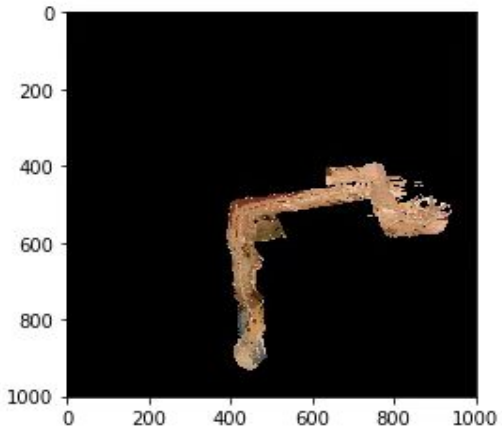

Fig. 24. Texture mapping of dataset 21

G. Discussion

1) SLAM part: AS we can see in above results, perform SLAM algorithm using dead-reckoning and particle filter have very samiliar results which make a lot of sense for the robot is on the same track during this whole time. BUt we can also see from the figure that the particle filter method do have a slightly better result. The map plotted by the particle filter have a more clear bound than the dead-reckoning, shown as below:

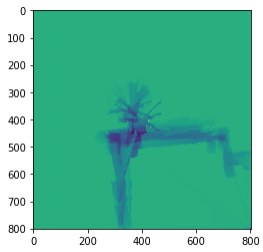

Fig. 26. Dead-reckoning map

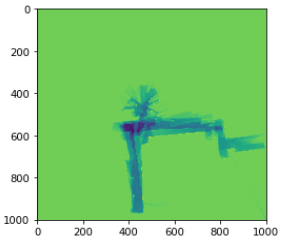

Fig. 27. particle filter map

The above result means that the position of the robot determined by particle filter is more accurate than the dead-reckoning.

There are still something did not work in this algorithm. For dataset 21, the map should be 'thinner' for there are some bumps on the trajectories of the robot so that the measurements are influenced. My algorithm did not sense that.

One thing I could try to improve my results is that I can enlarge the noise I put into the IMU measurement and encoder counts to make the particles more saperate and the result would be more accurate.

2) Texture mapping: There are still some flaws in my texture mapping. Besides of the basic color, my texture mapping also have some 'noise color' like the red point and color drawed on the map is kind blur, this could due to the thershold I set up is higher than it should be. Another problem is that my texture mapping did not color the initial spot the robot staied at. I could improve the algorithm base on this.