ECE 276 Project: Particle Filter SLAM

Chen Du
Department of Electrical Computer Engineering
University of California, San Diego
c6du@eng.ucsd.edu

***Abstract*—This paper presented the methods used in 2-D Simultaneous Localization & Mapping (SLAM) as well as texture mapping. In this project, I use particle filter to estimate the state of the robot and construct map, based on known inputs or observations (velocity, angle velocity and lidar ranges). For texture mapping part, I find the connections between world positions and camera pixels using the camera parameters and transformation matrices. I also use some techniques such as range weighted correlation and position shift to help refine the mapping result.**

Keywords—2D SLAM, Particle Filter, Texture Mapping

I. INTRODUCTION

Simultaneous Localization and Mapping is part of navigation problem for mobile platforms that involves capturing the data from sensors and simultaneously creating map and calculating its location in this map. SLAM can be used in autonomous vehicle which helps to tell where the obstacles are and avoid them. This can also help people estimate a map without doing it in person. This make it possible to use robot doing dangerous tasks instead of human. There are many ways to implement SLAM, mostly based on what kinds of sensors equipped by the robots.

In this project, based on the sensors we have (Encoders, Inertial Measurement Unit, 2D-Lidar, RGBD camera), I use the velocity, angular velocity and range information to realize 2D SLAM, and use RGBD camera data to do texture mapping based on the SLAM information. To estimate the states (position, orientation) of the robot car, I used particle filter and it gives quite good estimation results. Once I got the estimated position, I used this robot state and lidar data to update the 2D map. This method can give a pretty good result for SLAM.

After that, I also tried several techniques such as range weighted observation model, position shift to refine the mapping. And these techniques indeed help to get a much better performance.

In this paper, section II shows how I formulate the SLAM problem and the method I use for texture mapping. Section III tells the detailed technique approaches for the project. And section IV provides the results of the project.

II. PROBLEM FORMULATION

A. Localization

Based on the provided sensors on the robot car, we can get velocity information from encoder, angular velocity from IMU, range information from lidar.

The particle filter to can handle this case quite well (estimate the position, orientation). The particle filter usually has two steps: prediction and update. Suppose we have prior

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

x_t is the state at time t, $z_{0:t}$ is the observation from time 0 to t, $u_{0:t-1}$ is the input from time 0 to t-1, $\alpha_{t|t}^{(k)}$ is the weight for particle k, $\mu_{t|t}^{(k)}$ is the position for particle k, and

$$\delta\left(x_t; \mu_{t|t}^{(k)}\right) = \begin{cases} 1 & x_t = \mu_{t|t}^{(k)} \\ 0 & x_t \neq \mu_{t|t}^{(k)} \end{cases} \tag{2}$$

Then the prediction step is

$$\begin{aligned} p_{t+1|t}(x) &= \sum_{k=1}^{N_{t|t}} \alpha_{t|t}^{(k)} p_f\left(x \mid \mu_{t|t}^{(k)}, u_t\right) \\ &\approx \sum_{k=1}^{N_{t+1|t}} \alpha_{t+1|t}^{(k)} \delta\left(x; \mu_{t+1|t}^{(k)}\right) \end{aligned} \tag{3}$$

Usually, we need to do sampling based on the pdf and maintain the particle number.

For the update step, we update the weight based on the observation.

$$p_{t+1|t+1}(x) = \sum_{k=1}^{N_{t+1|t}} \left[\frac{\alpha_{t+1|t}^{(k)} p_h\left(z_{t+1} \mid \mu_{t+1|t}^{(k)}\right)}{\sum_{j=1}^{N_{t+1|t}} \alpha_{t+1|t}^{(j)} p_h\left(z_{t+1} \mid \mu_{t+1|t}^{(j)}\right)} \right] \delta\left(x; \mu_{t+1|t}^{(k)}\right) \tag{4}$$

and $p_h\left(z_{t+1} \mid \mu_{t+1|t}^{(k)}\right)$ is the observation model.

In this project, the state is the 2D position and orientation of the robot car.

$$state = \begin{pmatrix} x \\ y \\ \theta \end{pmatrix} \tag{5}$$

So, for prediction part, I use 2D discrete-time differential drive motion model. The model can be written as

$$x_{t+1} = f(x_t, u_t) := x_t + \tau \begin{pmatrix} v_t \sin c\left(\frac{\omega_t \tau}{2}\right) \cos\left(\theta_t + \frac{\omega_t \tau}{2}\right) \\ v_t \sin c\left(\frac{\omega_t \tau}{2}\right) \sin\left(\theta_t + \frac{\omega_t \tau}{2}\right) \\ \omega_t \end{pmatrix} \begin{matrix} \\ \\ \\ \end{matrix} (6)\, v_t$$

Where x_t is the state at time t, u_t is the known input (, ω_t), and τ is the time discretization. This is equivalent to p_f is a delta function. We don't use a continuous pdf p_f, instead, we add noise to the known input, this is much easier to realize, and can achieve similar results.

For update part, I follow the instructions and let

$$p_h\left(z_{t+1} \mid \mu_{t+1|t}^{(k)}, m\right) \propto \exp\left(\operatorname{corr}\left(y_{t+1}^{(k)}, m\right)\right) \tag{7}$$

The corr function calculate the similarity between the observed map (calculated by 2D lidar) and the known map. Since we'll need to update the weight later, we don't really need to normalize the observation model.

I later slightly change the corr function to the format of $\operatorname{corr}\left(y_{t+1}^{(k)}, m, range\right)$, which also consider the range between lidar and the obstacles. Since the obstacles of long range will be sparse. I consider the range information so that the correlation of longer distance obstacles will have higher weights. This modification helps to build a much accurate map.

B. Mapping

For mapping, I use occupancy grid mapping, which means the map is represented using a 2D matrix. The value for each cell shows the log-odds ratio. The map is updated using

$$\lambda\left(m_i \mid z_{0:t}, x_{0:t}\right) \coloneqq \lambda\left(m_i\right) + \sum_{s=0}^{t} \log g_h\left(z_s \mid m_i, x_s\right) \tag{8}$$

Since the map cells are assumed independent, I used a simpler model for $p_h\left(z_t \mid m_i, x_t\right)$ by specifying how much we trust the occupancy measurement of cell i:

$$g_h\left(1 \mid m_i, x_t\right) = \frac{p_h\left(z_t = 1 \mid m_i = 1, x_t\right)}{p_h\left(z_t = 1 \mid m_i = 0, x_t\right)} = \frac{80\%}{20\%} = 4 \tag{9}$$

$$g_h\left(0 \mid m_i, x_t\right) = \frac{1}{4} \tag{10}$$

C. Texture Mapping

For texture mapping, I use pinhole camera model.

Suppose we have an object at position $\left[X_w, Y_w, Z_w\right]^T$ in world frame. Using extrinsics

$$\begin{bmatrix} X_o \\ Y_o \\ Z_o \\ 1 \end{bmatrix} = \begin{bmatrix} R_{oc}R_{wc}^T & -R_{oc}R_{wc}^T p_{wc} \\ 0 & 1 \end{bmatrix} \begin{bmatrix} X_w \\ Y_w \\ Z_w \\ 1 \end{bmatrix} \tag{11}$$

And intrinsics

$$\begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = \begin{bmatrix} fs_u & fs_\theta & c_u \\ 0 & fs_v & c_v \\ 0 & 0 & 1 \end{bmatrix} \frac{1}{Z_o} \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} X_o \\ Y_o \\ Z_o \\ 1 \end{bmatrix} \tag{12}$$

We can have the pixel position of the object. Now, from RGBD sensor, we can calculate pixel position u, v and depth information Z_o. So, we can do an inverse process to get the world position of the specific pixel. Then threshold the height in world frame and extract the color value of the floor.

III. Technical Approach

A. Principle Formulation

Rotation matrix from body to world:

Suppose we have Roll(ψ), Pitch(θ), Yaw(ϕ) of the body, then the rotation matrix is

$$R_{B2W}\left(\psi, \theta, \phi\right) = R_z\left(\psi\right) R_y\left(\theta\right) R_x\left(\phi\right) \tag{13}$$

Where

$$R_z\left(\psi\right) = \begin{bmatrix} \cos\left(\psi\right) & -\sin\left(\psi\right) & 0 \\ \sin\left(\psi\right) & \cos\left(\psi\right) & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{14}$$

$$R_y\left(\theta\right) = \begin{bmatrix} \cos\left(\theta\right) & 0 & \sin\left(\theta\right) \\ 0 & 1 & 0 \\ -\sin\left(\theta\right) & 0 & \cos\left(\theta\right) \end{bmatrix} \tag{15}$$

$$R_x\left(\phi\right) = \begin{bmatrix} \cos\left(\psi\right) & -\sin\left(\psi\right) & 0 \\ \sin\left(\psi\right) & \cos\left(\psi\right) & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{16}$$

Transformation matrix from body to world:

The position of a point in world frame S_W can be calculated using

$$\begin{bmatrix} S_W \\ 1 \end{bmatrix} = T \begin{bmatrix} S_B \\ 1 \end{bmatrix} \tag{17}$$

Where S_B is the position in robot body frame and

$$T = \begin{bmatrix} R_{B2W} & p_{B2W} \\ 0 & 1 \end{bmatrix} \tag{18}$$

is the transformation matrix.

B. Data preprocessing

I inspected the data first and found that angular velocity is noisy and has a higher sampling frequency. So before the SLAM process, I did a low pass filter with cutoff frequency 10Hz to the IMU data. This helps to improve the performance of the SLAM result.

C. SLAM

Map Initialization

Before the Localization and Mapping, we first need to build a map as reference. So, we suppose the starting point of the robot is the origin and use the first scan of the lidar information to build the initial map. We first make a map with all points having zeros log-odds ratio.

From lidar dataset, we can use the angle and range information to calculate the positions of each point obstacles in lidar body frame. After that, using the transformation matrix, we can calculate these positions in robot body frame, and then again, the positions in world frame. Based on the boarder coordinates and resolution of the map, we can calculate the cells these points fell in, using:

$$i_{x-cell} = \left\lceil \frac{x - b_{x\min}}{res_x} \right\rceil - 1 \tag{19}$$

$$i_{y-cell} = \left\lceil \frac{y - b_{y\min}}{res_y} \right\rceil - 1 \tag{20}$$

i_{x-cell} is the index of x coordinate of point in the map, x is the x position of the point, $b_{x\min}$ is the minimum boarder position of the map, and res_x is the resolution along x axis of the map. Similar representation for y axis.

Once we get the positions of the obstacles and the position of lidar (using same transformation process), we can draw an area with vertices of these points (cv2.drawcontour). The cells in the area are regarded as free, and vertices except lidar are regarded as walls. Finally, using the occupancy grid mapping model mentioned in section II, we add log (4) to cells which are walls and add minus log (4) to cells which are free. To show whether the cell is occupied or not, we can set a threshold and every log-odd ratio value larger than it will be regarded as occupied. Usually I set the threshold being 2 * log (4) for occupied and -2 * log (4) for free, the value between these two are regarded as unknown area.

Time Stamps

During the SLAM process, I decide to go through the encoder data and use it as main time sequence. For other sensors, I choose the data that has the closest time stamp to current encoder time stamp as the input or observation. This is because I think localization would be the most import thing for the map construction, and I don't want to lost information about how long the robot moves.

Localization and Particle Filter

From section II, the prediction step is function (3). Usually the pdf $p_f\left(x \mid \mu_{t|t}^{(k)}, u_t\right)$ is a continuous function, and we need to sample N particles using the probability function. In this project I set N=200. But in this case, we use the discrete-time motion model of formula (6) to do prediction, instead of using the continuous pdf, we assume it being a delta function. We add Gaussian noise to the input (v , ω) to implement the uncertainty. So, we don't need to do sampling to maintain the particle number but get a similar result. The Gaussian standard deviation is set to be 0.5 times current input value.

Since the angular velocity data is noisy, I calculate the mean of two IMU data which are closest to the current time stamp as the input angular velocity.

For update step, we use the observation model of function (7), this requires a correlation function $\text{corr}\left(y_{t+1}^{(k)}, m\right)$. Given a list of cell locations in map, the correlation function calculates how many occupied cells located in the listed positions.

$$\text{corr}\left(y_t^{(k)}, m\right) = \sum_{i=1}^{N_{angle}} \mathbf{1}\left(m\left(y_{t;i}^{(k)}\right), occupied\right) \tag{21}$$

Where $y_{t+1;i}^{(k)}$ is the i_{th} angle point in particle k at time t, and

$$\mathbf{1}(a,b) = \begin{cases} 1 & a = b \\ 0 & a \neq b \end{cases} \tag{22}$$

Since a better prediction should fit more to the known map, this correlation will reflect which particle has the best position at this prediction step. However, in this project, I not only estimate the correlation between the map and the exact detected cells, but also the cells around the detected cells with an area of 9*9 square. And use the best correlation value in this square as the result of this particle. This would help avoid dropping particles that have lower value because of quantization shift. Besides the position of the best correlation value in the 9*9 square will also help to tell how this particle can be shifted to a better position. After calculating the correlation for each particle, we can find the best particle, which will be used for map update, as well as the best shift position for each particle.

Using formula (4) and (7), we can update the weight for each particle.

$$\begin{aligned} \alpha_{t+1|t+1}^{(k)} &= \frac{\alpha_{t+1|t}^{(k)} p_h\left(z_{t+1} \mid \mu_{t+1|t}^{(k)}\right)}{\sum_{j=1}^{N_{t+1|t}} \alpha_{t+1|t}^{(j)} p_h\left(z_{t+1} \mid \mu_{t+1|t}^{(j)}\right)} \\ &= \frac{\alpha_{t+1|t}^{(k)} \exp\left(\sum_{i=1}^{N_{angle}} \mathbf{1}\left(m\left(y_{t;i}^{(k)}\right), occupied\right)\right)}{\sum_{j=1}^{N_{t+1|t}} \alpha_{t+1|t}^{(j)} \exp\left(\sum_{i=1}^{N_{angle}} \mathbf{1}\left(m\left(y_{t;i}^{(j)}\right), occupied\right)\right)} \end{aligned} \tag{23}$$

Since we also have the best shift position for each particle, I also shift the position of each particle based on this information.

I later changed the correlation function to

$$\text{corr}\left(y_t^{(k)}, m, range_t^{(k)}\right) = \sum_{i=1}^{N_{angle}} \left(range_{t;i}^{(k)}\right)^{\gamma} \mathbf{1}\left(m\left(y_{t;i}^{(k)}\right), occupied\right) \tag{24}$$

I take range information into consideration. This is because I find the detected points with long distance are much sparser, and it's less likely to get correlation value from these points. But when we use the particle shift, since the closer observed occupied points are denser. The previous correlation function would let the points keeps shifting backwards because it can get a higher correlation value. After adding ranges as a weight into the function, even when it has correlation with few long-range points, it would get a high reward, this would help it to shift to the correct position in stead of going back. The results shows that the introduction of range increase the accuracy significantly. For order γ value I set it to 2, and it gives quite good result.

Besides, if the condition (25) is satisfied. I do a stratified resampling. Where $N_{threshold}$ is set to be 0.75 times particle number.

$$N_{eff} = \frac{1}{\sum_{k=1}^{N_{t|t}} \left(\alpha_{t|t}^{(k)}\right)^2} \leq N_{threshold} \tag{25}$$

The pseudocode of the stratified resampling is showed as follows:

1 : *Input* : *particle set* $\left\{\mu^{(k)}, \alpha^{(k)}\right\}_{k=1}^{N}$
2 : *Output* : *resampled particle set*
3 : $j \leftarrow 1, c \leftarrow \alpha^{(1)}$
4 : *for* $k = 1, ..., N$ **do**
5 : $u \sim U(0, 1/N)$
6 : $\beta = u + (k-1)/N$
7 : *while* $\beta > c$ **do**
8 : $j = j + 1, c = c + \alpha^{(j)}$
9 : *add* $\left(\mu^{(j)}, 1/N\right)$ *to the new set*

Mapping

From localization part, we can get the updated weights as well as shift positions calculated by the correlation part for each particle. I choose the particle with the best weight as the best estimated position. And use its range information to update the occupancy grid map. Before doing the map update, we need to shift the position of the detected points due to the shift of particle. The map update process is the same as in map initialization part.

I do localization part and mapping part alternatively to realize the SLAM.

Texture Mapping

In RGBD camera data, we can have RGB image and disparity matrix. Using the instruction, we can calculate the depth information and connection between depth information and its position in RGB image

$$dd = \left(-0.00304 * d + 3.31\right)$$

$$depth = 1.03 / dd$$

$$rgbi = \frac{i * 526.37 + dd * \left(-4.5 * 1750.46\right) + 19276.0}{585.051} \quad (26)$$

$$rgbj = \frac{j * 526.37 + 16662.0}{585.051}$$

Where d is the value in disparity matrix, and rgbi, rgbj shows the pixel position in RGB image. I eliminate the points that has negative depth which are not valid.

We now have pixel positions u, v and depth Z_o. And we need to estimate their positions in world frame. From (12) we can have

$$\begin{bmatrix} X_o \\ Y_o \\ Z_o \end{bmatrix} = Z_o \begin{bmatrix} fs_u & fs_\theta & c_u \\ 0 & fs_v & c_v \\ 0 & 0 & 1 \end{bmatrix}^{-1} \begin{bmatrix} u \\ v \\ 1 \end{bmatrix} \quad (27)$$

Where f and c are parameters of the camera which are already known.

Then we use (11) to get points in robot body coordinates

$$\begin{bmatrix} X_B \\ Y_B \\ Z_B \\ 1 \end{bmatrix} = \begin{bmatrix} R_{oc} R_{C2B}^T & -R_{oc} R_{C2B}^T p_{C2B} \\ 0 & 1 \end{bmatrix}^{-1} \begin{bmatrix} X_o \\ Y_o \\ Z_o \\ 1 \end{bmatrix} \quad (28)$$

Where C2B means camera to robot body

$$R_{oc} = \begin{bmatrix} 0 & -1 & 0 \\ 0 & 0 & -1 \\ 1 & 0 & 0 \end{bmatrix} \quad (29)$$

Then again, we use the transformation matrix to convert to world frame

$$\begin{bmatrix} X_W \\ Y_W \\ Z_W \\ 1 \end{bmatrix} = \begin{bmatrix} R_{B2W} & p_{B2W} \\ 0 & 1 \end{bmatrix} \begin{bmatrix} X_B \\ Y_B \\ Z_B \\ 1 \end{bmatrix} \quad (30)$$

Doing this, we find the connections between pixel position in RGB image and the world position. Because we want to do texture mapping for the floor, I set two thresholds and height between -0.5m to 0.25m will be preserved. Then I color the 2D map at $\left[X_W, Y_W\right]^T$ with pixel value at image position $\left[u, v\right]^T$. To have a better show, I implement texture during the SLAM process, so that I can get area of free at each time stamp and use it as a mask for texture mapping.

IV. RESULTS

I optimize the correlation function using numpy operation instead of for loop. It takes about 1500s to go through a dataset with all techniques implemented. The map resolution is set to 0.05m while the boarder positions are set based on the datasets range. I also provide 3 video files in programming zip files, showing how map is constructed for test20, test21, and test23.

The followings are some interesting results generated using different techniques. The black part as unknown area, gray part are free area and white part are occupied area (wall).

A. Result without particle shift

This part shows the result of original method, without particle shift or range weighted correlation.

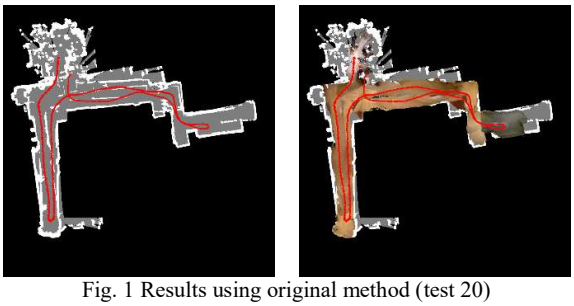


Fig. 1 Results using original method (test 20)

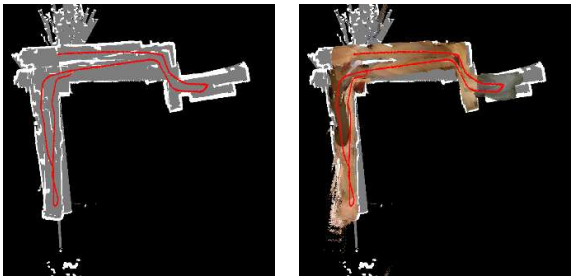


Fig. 2 Results using original method (test 21)

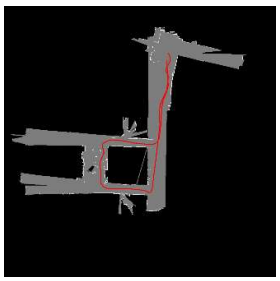


Fig. 3 Results using original method (test 23)

We can notice the original method can give a relatively good result, we can find the basic structure of the building. However, it's also obvious that the path is not well estimated, which directly result in the inaccuracy of the mapping result. But for test23, the original provided the best results compared to other techniques.

B. Result with particle shift

This part shows the result of the method using particle shift but not range weighted correlation. We can notice, compared to the original method, adding particle shift helps get a much better result. However, the closer points are denser than distant points, this would let the particle keeps shifting back. This is extremely obvious in test 21, the top right part of the free area is not passed through by the robot, compared with the next part which use all the techniques. If we look at the trajectory, despite the wrong particle shift, the trajectory is much better than the previous approach. For test 20, this approach could get a well result.

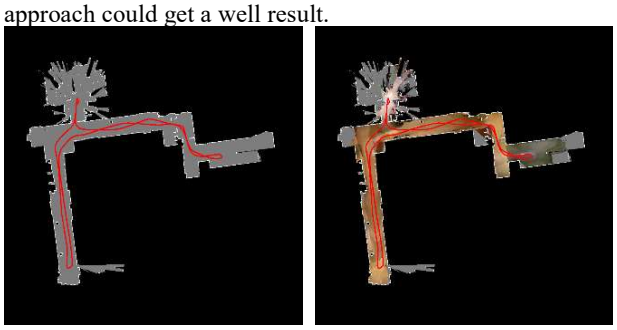
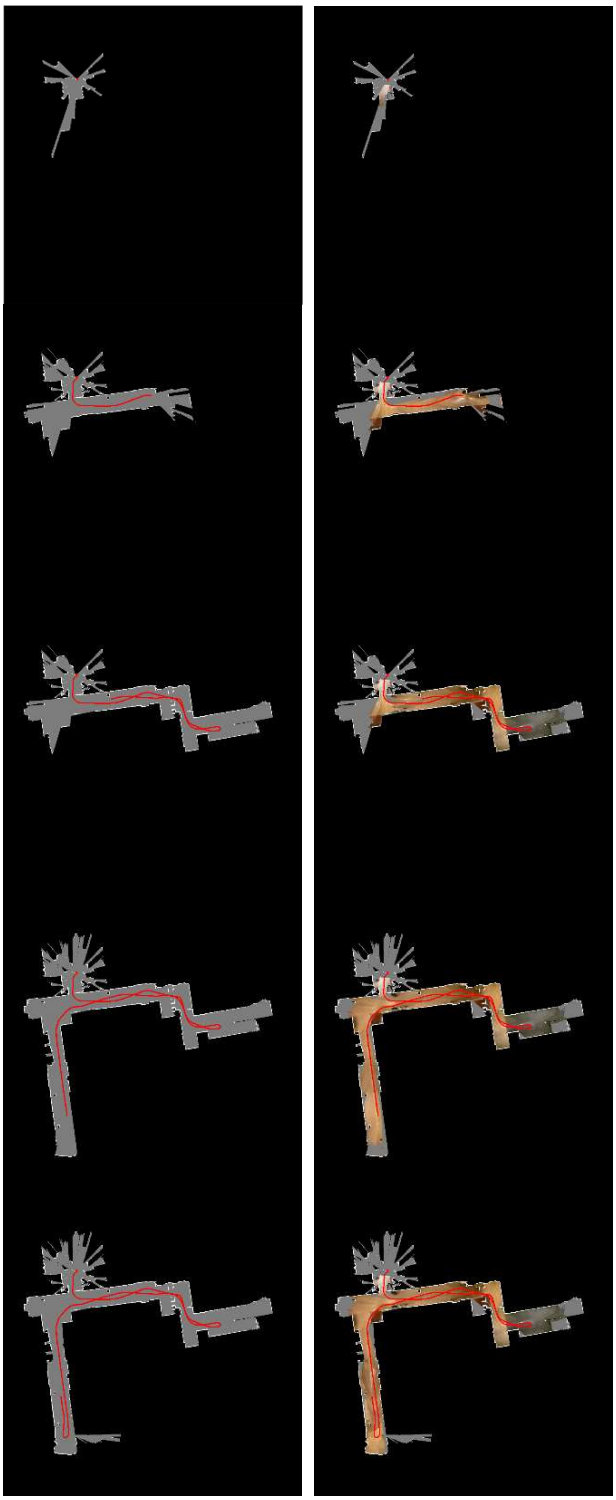


Fig. 4 Results using particle shift (test 20)

Fig. 5 Results using particle shift (test 21)

C. Result with particle shift and range weighted correlation

This part shows the result of the method using particle shift and range weighted correlation. We can find that this method can get excellent result for test 20 and test 21. However, these techniques are not good enough to provide good results for test 23. It's obvious that there are still shifting back at the beginning of the trajectory, and wrong angle estimation at the last turn. The shifting method is affecting the performance of SLAM. Adding more particles and more noises may solve this problem. But since it takes too much time to go through a test set, I didn't really try this.



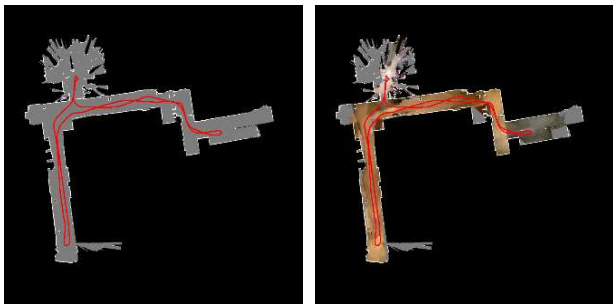


Fig. 6 Results using all techniques (test 20)

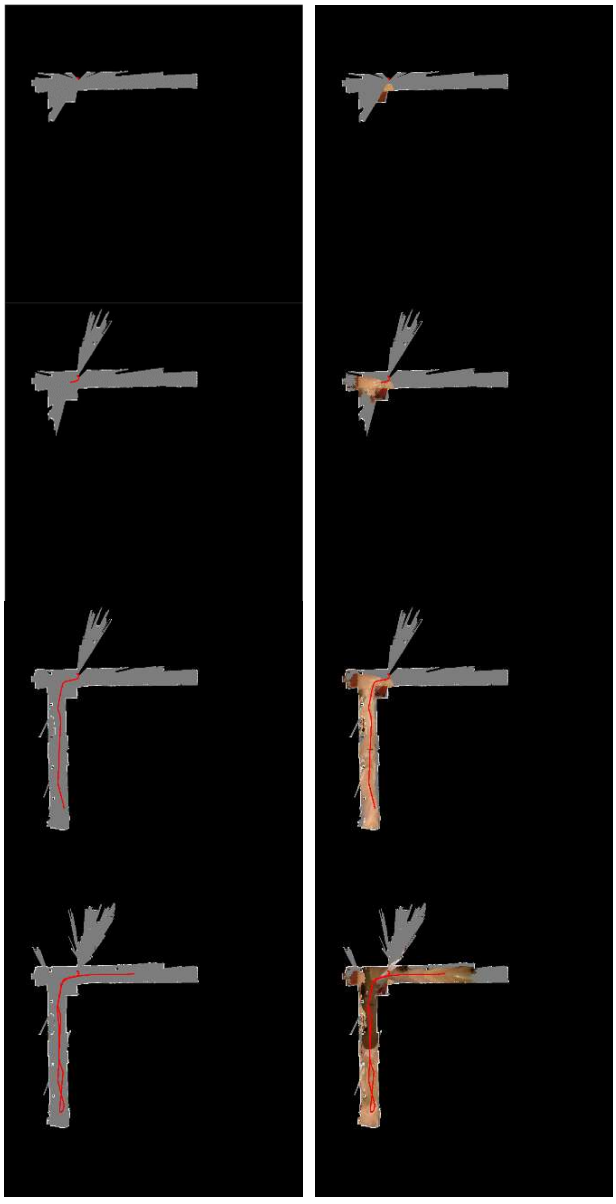


Fig. 7 Results using all techniques (test 21)

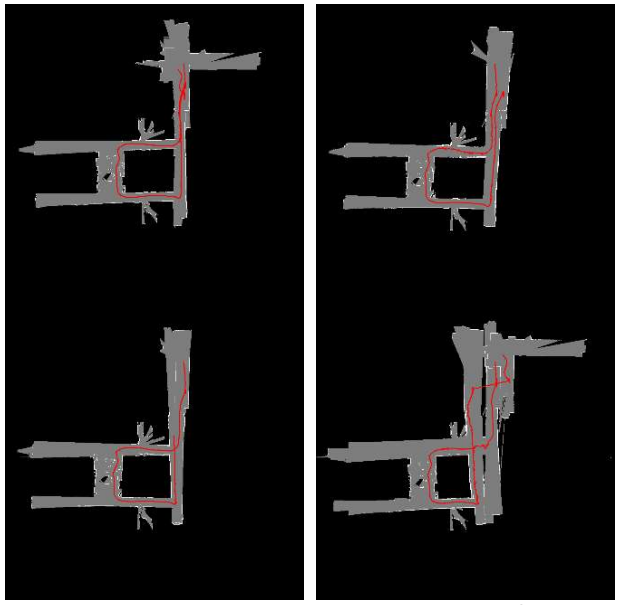


Fig. 8 Results using all techniques (test 23, $\gamma = 2$)