

Simultaneous Localization and Mapping

Yidi Zhu

Department of Electrical Computer Engineering

University of California, San Diego

yizhu@eng.ucsd.edu

Abstract—This paper presented approaches using Softmax to do color segmentation and measuring score of “barrels” to draw bounding boxes.

Index Terms—Softmax, Image Processing

I. Introduction

In robotics and mapping area, Simultaneous localization and mapping (SLAM) is an important way to construct a map of the unknown environment and keep tracking where the robot is. This chicken and egg type problem could be solved by SLAM in a easily understand form. Such computational way is implemented many kinds of intelligent devices such as cars, detecting machines ad so on.

In this project, there are four steps, Prediction, Update, Mapping and Texture. Predication and update could be combined to one step to get the current position of the robot. By transforming the LIDAR scan to the world frame, the map of the space can be constructed at this timestamp. At the same time, with the robot’s location (best particle position) and the RGBD data, the occupancy grid map can be colored by floor color. With iteration of above four steps, we could construct the map of the space.

II. Problem Formulation

① Rotation

The most important method of SLAM is rotation. Not only in the update step and the mapping step, we should transform the lidar into the world frame to compare to or build the

map.

	Rotation $SO(3)$	Pose $SE(3)$
Representation	$R : \begin{cases} R^T R = I \\ \det(R) = 1 \end{cases}$	$T = \begin{bmatrix} R & p \\ 0 & 1 \end{bmatrix}$
Transformation	$s_W = R s_B$	$s_W = R s_B + p$
Inverse	$R^{-1} = R^T$	$T^{-1} = \begin{bmatrix} R^T & -R^T p \\ 0 & 1 \end{bmatrix}$
Composition	${}_W R_{t_k} = {}_W R_{t_0} \prod_{i=0}^{k-1} {}_{t_i} R_{t_{i+1}}$	${}_W T_{t_k} = {}_W T_{t_0} \prod_{i=0}^{k-1} {}_{t_i} T_{t_{i+1}}$

Fig.1 Rotation and Pose in 3d

We use the showing formula to do the transformation. The left contains only rotation and the right one contains both rotation and shift.

② Particle Filter

Particle Filter contains 2 steps:

I. Prediction:

$$\mu_{t+1|t}^{(k)} = f\left(\mu_{t|t}^{(k)}, u_t + \epsilon_t\right)$$

The function f is the motion model in this case which is:

Discrete-time model with time discretization τ :

$$s_{t+1} = f(s_t, u_t) := s_t + \tau \begin{pmatrix} v_t \text{sinc}\left(\frac{\omega_t \tau}{2}\right) \cos\left(\theta_t + \frac{\omega_t \tau}{2}\right) \\ v_t \text{sinc}\left(\frac{\omega_t \tau}{2}\right) \sin\left(\theta_t + \frac{\omega_t \tau}{2}\right) \\ \omega_t \end{pmatrix}$$

II. Updating

$$p_h(z | x, m) = \frac{e^{\text{corr}(y, m)}}{\sum_v e^{\text{corr}(v, m)}} \propto e^{\text{corr}(y, m)}$$

The observation model depends on the Map Correlation function. In other words, the probability of a particle being correct is proportional to the correlation between map and the lidar scan.

III. Resampling

By using the particle filter, sometimes the

particles will be diming because of very small weight, and most weight accumulate on a few particles; thus, the resampling step is need, to redistribute the weight on particles.

Stratified (low variance) resampling

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

Fig.2 The principle of Stratified resampling

③ Mapping

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

So we can using the particle position, angles to transform the scanning data lidar frame to world frame and operate the log_odd map(increase the value of an occupied grid and otherwise decrease it). Finally, we could set a threshold to determine whether the grid is unknown or occupied or not occupied.

④ Texture

► **Extrinsics:**

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

► **Projection and Intrinsics:**

$$\begin{pmatrix} u \\ v \\ 1 \end{pmatrix} = \underbrace{\begin{bmatrix} f s_u & f s_v & c_u \\ 0 & f s_v & c_v \\ 0 & 0 & 1 \end{bmatrix}}_{\text{calibration: } K} \underbrace{\frac{1}{Z_o} \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix}}_{\text{canonical projection: } \Pi_0} \begin{pmatrix} X_o \\ Y_o \\ Z_o \\ 1 \end{pmatrix}$$

pixels

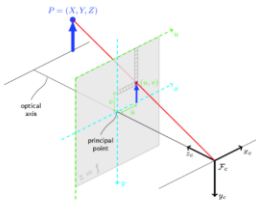


Fig.3 The principle of pinhole camera

We should use the knowledge of camera model to transform the 2d image into the camera frame(use the disparity image)

III. Technical Approach

I separate my approach into 3 process.

1) Pre-processing

Before we begin the process of SLAM. We have several pre tasks.

- I. Load the data
- II. Initial the Map and the particles
- III. Applying a low-pass filter to reject noise in IMU data(since this data is full of small high frequency noise, this operation could make it smoother)
- IV. Match the data between encoder and IMU since the time stamps of them is not the same

2) SLAM

In SLAM, I make an order of prediction, mapping, updating and resampling (I will explain for this.)

I. Prediction

Reading the data of encoder to get the traveled distance (just the average of all the four wheels) of the robot and the data of IMU (only use the raw rate) to get the angular velocity around the z-axis. Then use this information to predict the next state of the robot.

Advanced processing: add a $\mu = 0$, $\sigma = 0.095$ Gaussian noise to the position of every particle. (No noise in the angular of the particle since we will introduce the variance of angular in updating.)



Fig4. Trajectory (prediction only) without and with noise

For dataset20, we can predict the trajectory of the robot without the mapping and updating. We can see that with a noise on both position and angular, the trajectory will have some variation from the raw one. However, if we only change the angular by accumulating the angular velocity, the noise on angular could not fixed. Then we consider the angular data as noisy (do not make it noisier) and add a for loop in updating to add some variation in the angular and then pick the angular with the largest correlation value. (This would slow down the process but much more accurate.)

II. Mapping

Transforming the points which detected by the LIDAR sensor into the world frame. Noticing that the points detected is the occupied grid and we also need the free grid, we use the bresenham2D function to get the points between the start point and the end.

Then increasing the log_odd for the occupied grid and decrease the log_odd for the free grid (set the gh to 4 and constrain the log_odd to avoid overconfident).

Finally, we can get the map(contains -1,0,1) form the log_odds.

*Why put mapping at this place? For example, if we are at a long corridor, the MapCorrelation would be better at previous points since the map there has the same shape and a high confidence. So, constructing the map before updating could release the risk of unstoppable shift.

III. Updating

In updating, we both update the parameter

of the particle and the weight of them.

1) Parameter

By using the mapCorrelation function, we could get the correlation between the Lidar Scan and the existing map. This approach is to modify the small shift caused by the noised data of the velocity.

Noticing that we cannot modify the angular by the mapCorrelation because we have decided the transformation before introducing a range matrix. However, we could add a variation before deciding the transformation, so we could test which angular could reach a max correlation.

Then we could do mapCorrelation in the direction picked in several small different angular. For every particle, shifting it to the position of the largest correlation value.

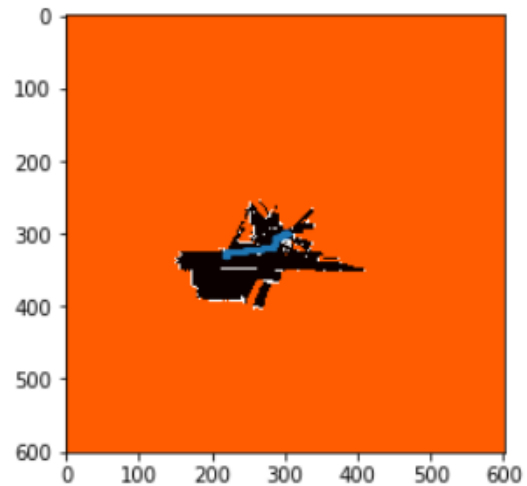


Fig.5 The error caused by the given Correlation function

*: The given mapCorrelatino function would cause some problems like the unstoppable shifting of the particle. This is because the sentence:

```
ix = np.int16(np.round((y1-ymin)/yresolution))
```

It is different from the sentence by which we transform the meter into cell:

```
ix = np.ceil...
```

And it may lead to a situation that the left cell at the left of the origin will have a high frequency to obtain a larger correlation value. Especially at a long corridor, the particle will continue shifting left until exceed the left bound.

2) Weight

Finally, use the correlation value of all particles to do SoftMax and re-weighting the particles.

IV. Resampling

We use the stratified resampling to resample the particles when the effective Number of the particle is lower than the threshold (set it to 5 when $N=100$).

When doing stratified resampling, we initial the weight all to $1/N$.

```

4: for  $k = 1, \dots, N$  do
5:    $u \sim \mathcal{U}(0, \frac{1}{N})$ 
6:    $\beta = u + \frac{k-1}{N}$ 
7:   while  $\beta > c$  do
8:      $j = j + 1, c = c + \alpha^{(j)}$ 

```

From the Fig.2, we could infer that samples with large weights would be chosen at least once and the samples with small weight would be chosen at most once. As a result, it is feasible resample to reserve the distribution and prevent the weight to focus on only one particle.

V. Texture

To paint the floor, we need to turn the picture frame into the camera frame.

Noticing that in the steps before, we consider the problem as a 2D problem, however,

we need to solve it in 3D space. It is a must to consider the height of all the objects.

Furthermore, the depth camera and rgb camera are not in the same location (there is an x-axis offset between them) and hence it is necessary to use a transformation to match the color to the depth. We just reshape the 2d pixels into an array and adding the depth into it.

VI. Speed Up

I find several following ways to speed up the programming.

- 1) Use a small range to do mapCorrelation. Since the frequency of updating is greatly larger than the moving speed, we could introduce a small correlation to speed up.
- 2) Vectorize the solution of texture: Adding the depth into the matrix to do vector manipulation.
- 3) Use fewer particles: since this a SLAM problem, we only need to concern the area around the particle in a small range. We do not need to introduce too many particles.

IV. Results:

dataset:20

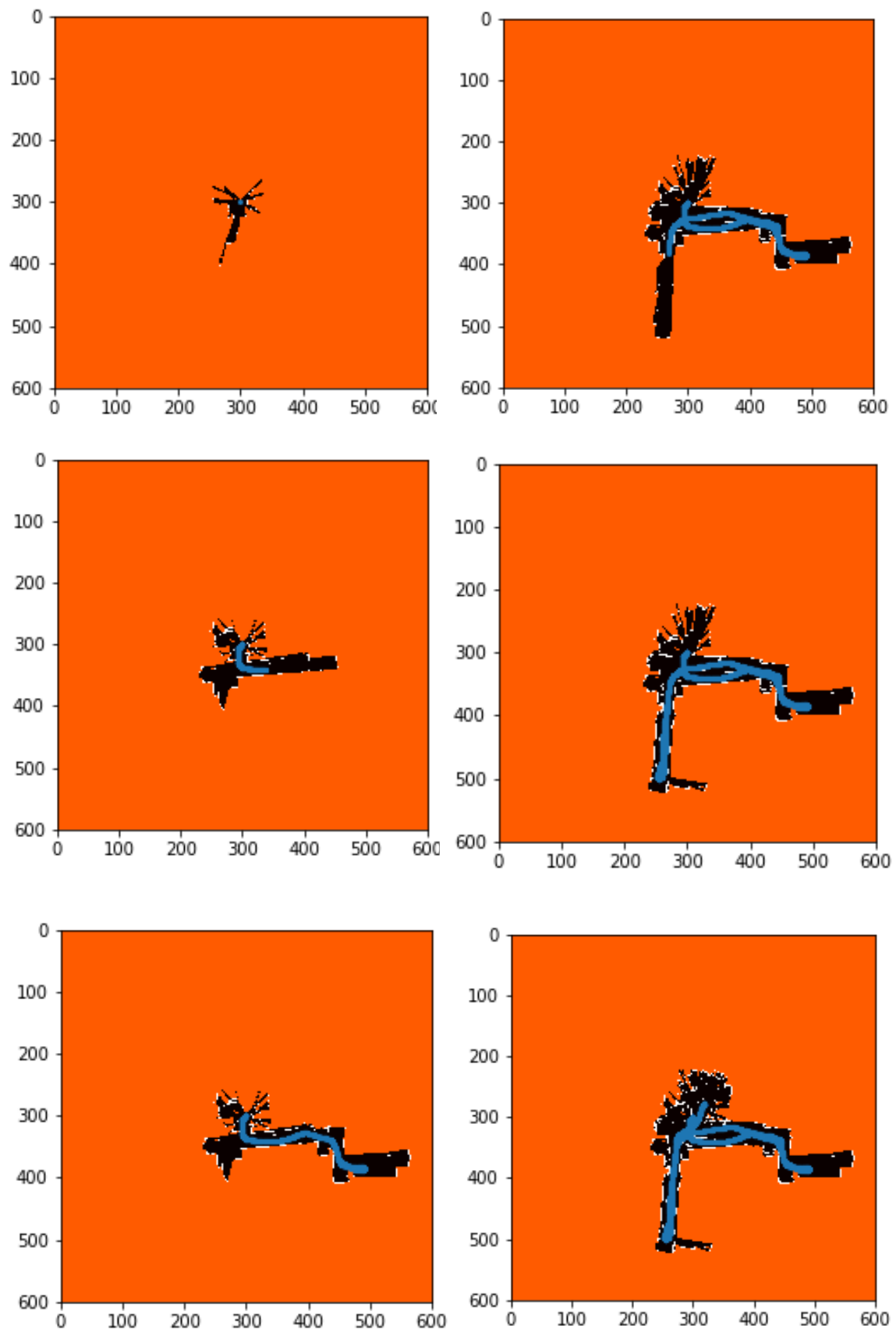


Fig.6 SLAM of dataset 20

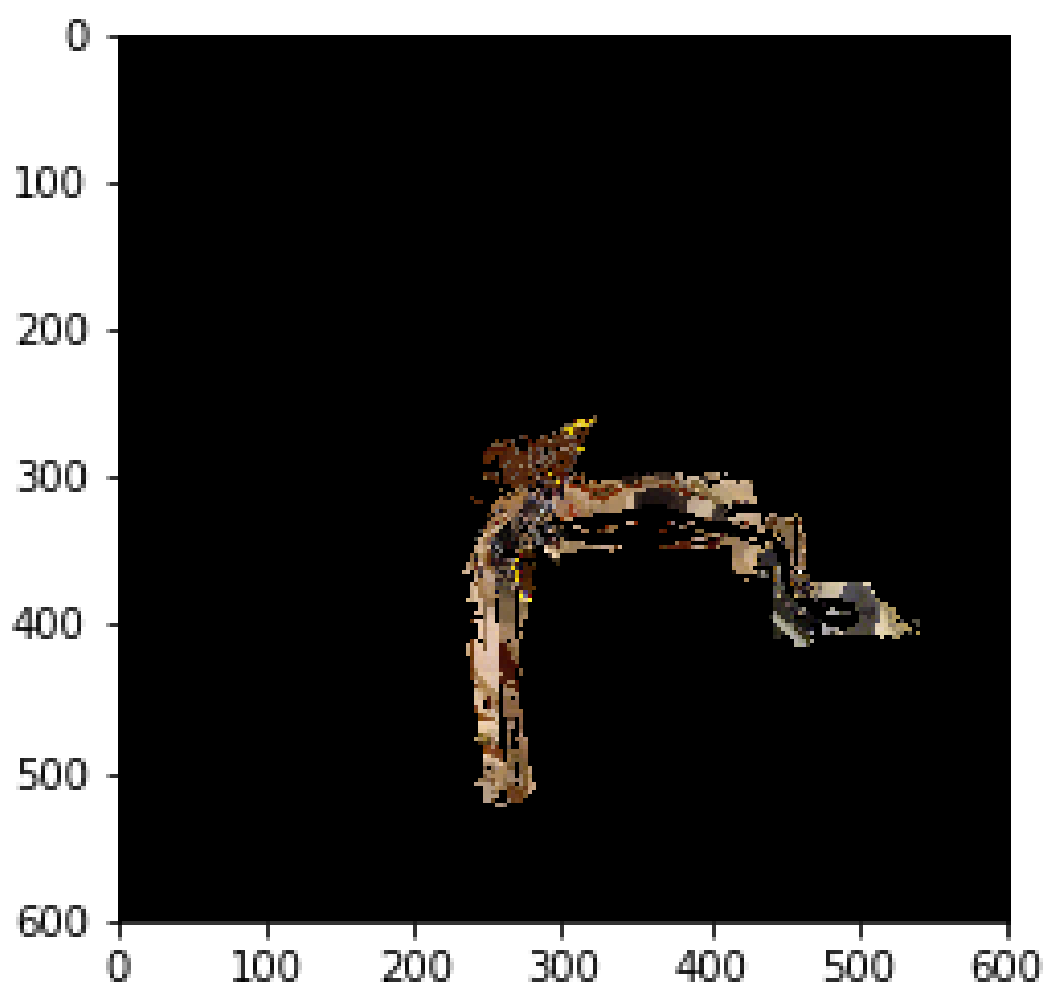


Fig.7 Texture of dataset 20

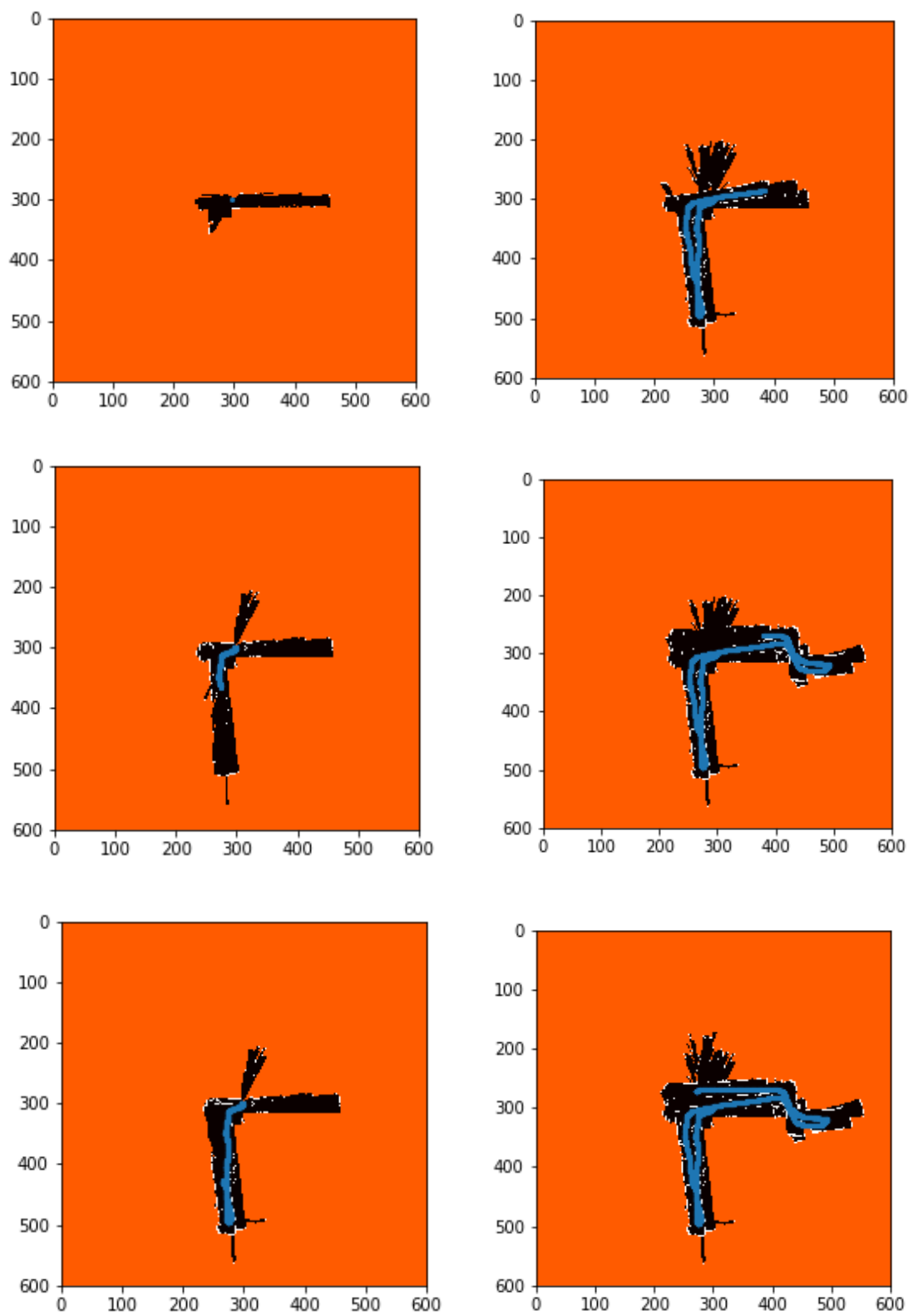


Fig.8 SLAM of dataset 21

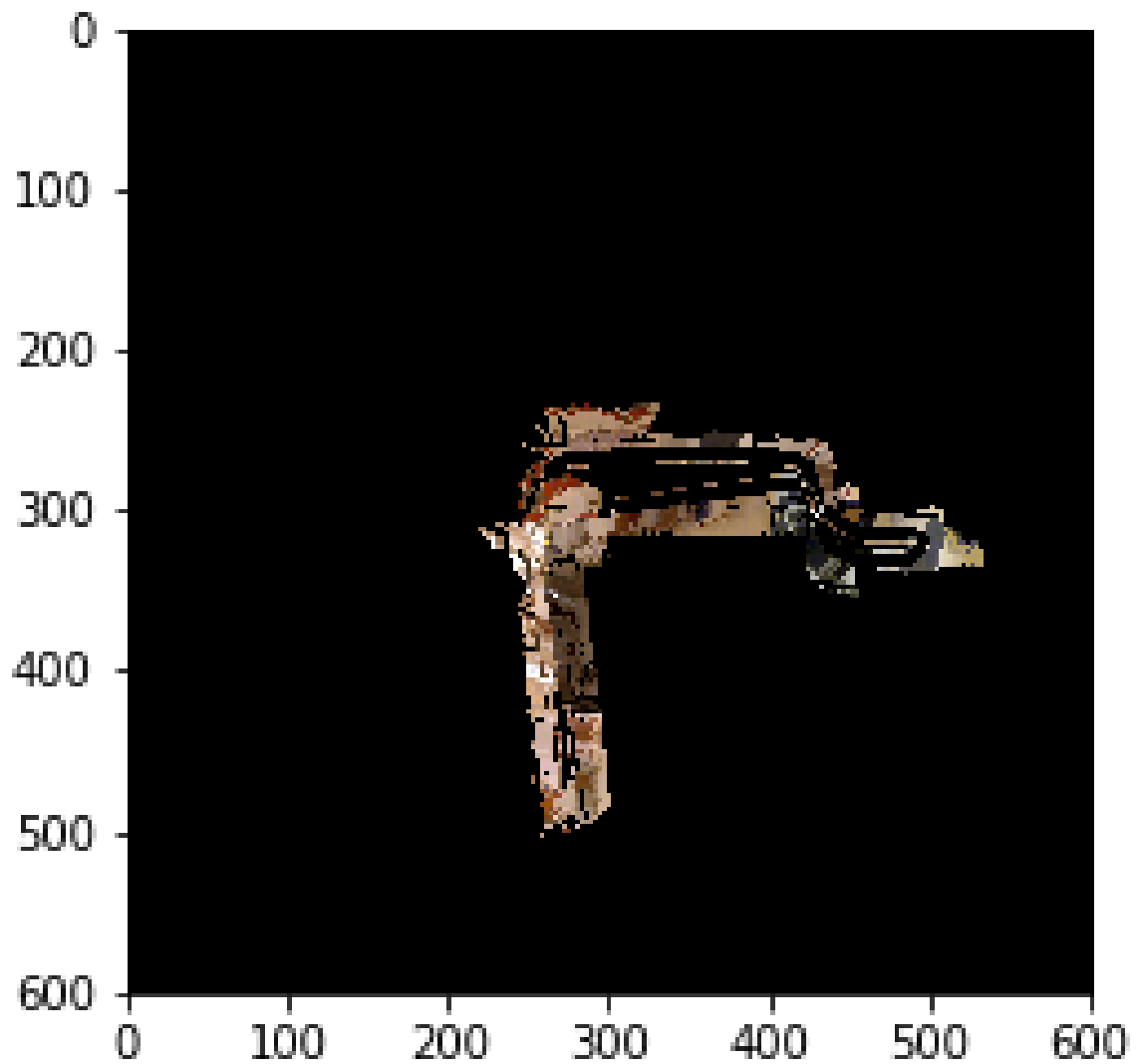


Fig.9 Texture of dataset 21

We can see that the SLAM worked for dataset 20 and 21. The map has little variation and the trajectory could fit the previous route.

However, the texture of both pictures does not look good, the transformation is not accurate enough to project the pixel frame into the world frame. I think I should check the parameters of the transformation and the setting of the threshold to make it much more accurate.

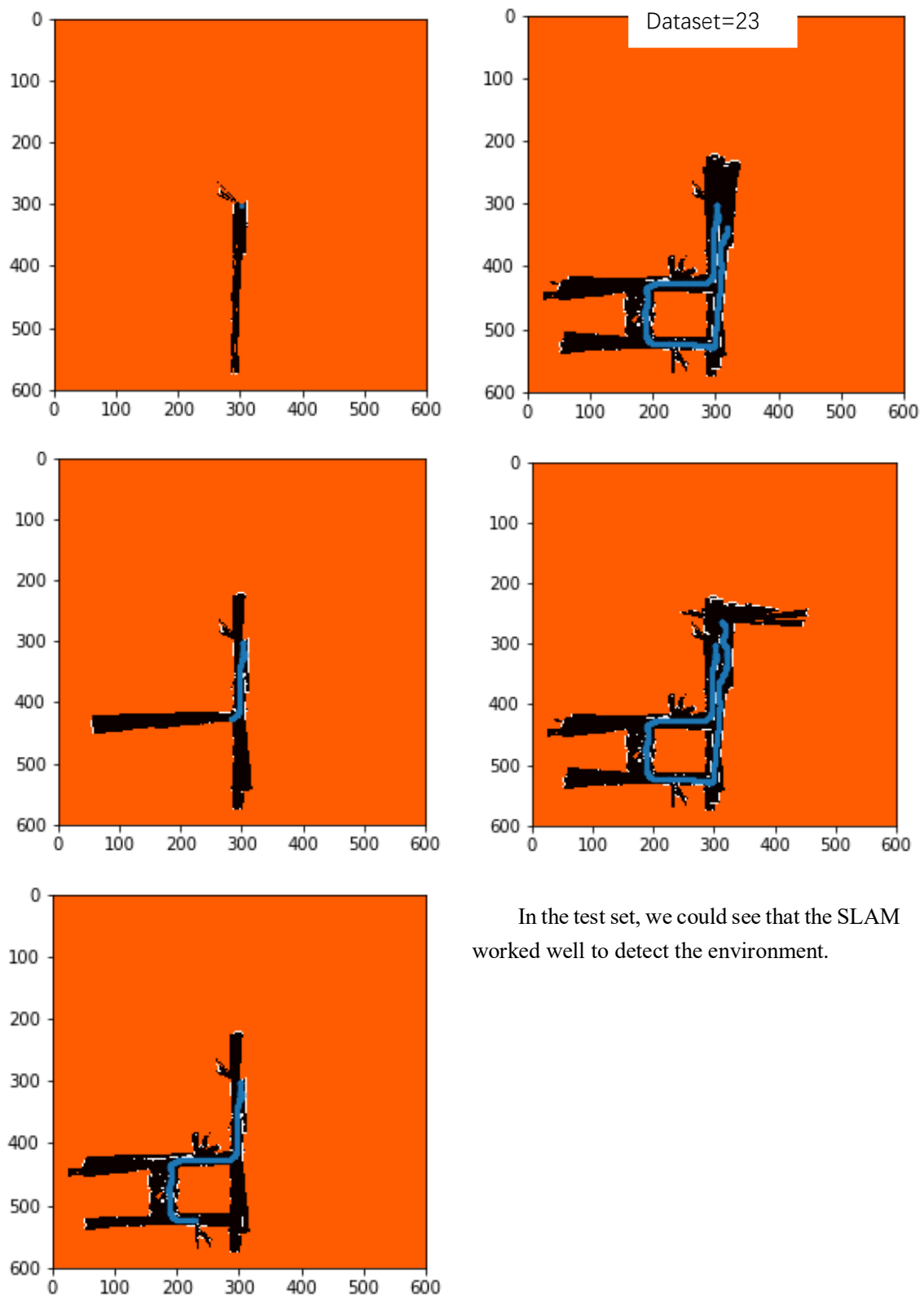


Fig.10 SLAM of dataset 23