Optimizations on GMapping SLAM

Abstract-- Recent years, GMapping has been one of the most widely-used open source projects to solve the LiDAR-based 2D simultaneous localization and mapping(SLAM) problem. This approach uses a particle filter in which each particle carries an individual map of the environment and the Robot pose. In this paper, we present adaptive techniques for optimizing this project in three aspects, that are motion model, sensor data preprocessing, and mapping.
Index Terms-- GMapping, sample motion model, occupancy grid map, TSDF

1. **Introduction**

SLAM is one of the fundamental tasks of mobile robots, it is considered to be a complex problem because for localization a robot needs a consistent map and for acquiring a map the robot requires a good estimate of its location. This mutual dependency between the pose and the map estimates makes the SLAM problem hard(Grisetti et al., 2007).

The sensor in this project is laser range finder and inertial measurement unit(IMU). Laser range finder is one of the most widely used sensors for the indoor SLAM, it measures the range to nearby objects, distance and orientation may be measured along a beam. IMU is an inertial sensor, it measures motion like velocity, acceleration, angular velocity and angular acceleration, it is usually as a supplement to or in place of odometry.

2. **GMapping Architecture**

GMapping was developed in 2007, and it is still one of the most common systems for robot application(Filipenko et al., 2018). It is based on Rao-Blackwellized particle filter to solve the SLAM problem, it is a complete system, so it was taken as the basis for my thesis experiment.

In GMapping, Robot Operating System(ROS) is used for providing peer to peer communication for the network of software modules, it provides raw sensor data including IMU and LaserScan ROS messages, and the processing pipeline in Figure 1 starts by receiving IMU and LaserScan measurements. In the “Init” process, the map coordinates set as (0,0), and a batch of particles will be initialized. Next, time alignment and spatial calibration will be done in “Data preprocessing”. Before each iteration of the filter and for each particle, the probability $p(x_t|x_{t-1}, u_t)$ is based on the motion model to get a new pose hypothesis x_t. Combine the new pose and new measurements, particles can be scored and get an optimal particle. The optimal particle carries the robot pose and map, in such conditions the map can be extended with the known robot pose.

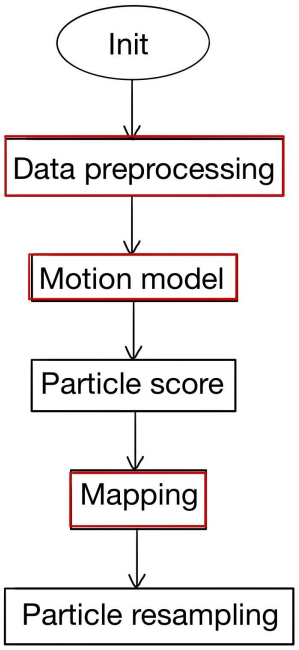


Figure 1, the pipeline of GMapping project

This thesis focuses on modifying the GMapping SLAM in three aspects, that are Data preprocessing, Motion model, and Mapping. It is organized as follows: After introducing the related technologies in these three fields, we apply them to experiments and show the results.

3. Related Technologies

3.1 Sensor data preprocessing

The dataset in this project is acquired with a 2-D laser range finder, as we know that this kind of sensor is rotationally measured, so each of the point data in one scan is not measured at the same time but sequentially. A laser scan might be distorted when a rangefinder moves during scanning. Figure 2 explains how the scan is distorted while a rangefinder moves. A black line in Figure 2(a) represents an environment and the rangefinder starts moving in a direction indicated by the arrow. Blue points in Figure 2(b) represent a raw scan data. Note that it is distorted because a scanning component rotates in a counter-clockwise direction during measurements. If we try to estimate the transformation without compensating the distortion, a wrong transformation will be estimated as shown in Figure 2(c). By correcting the scan, we obtain rectified data(red points in Figure 2(b)). Finally, a more accurate transformation is estimated from rectified data as shown in Figure 2(d).

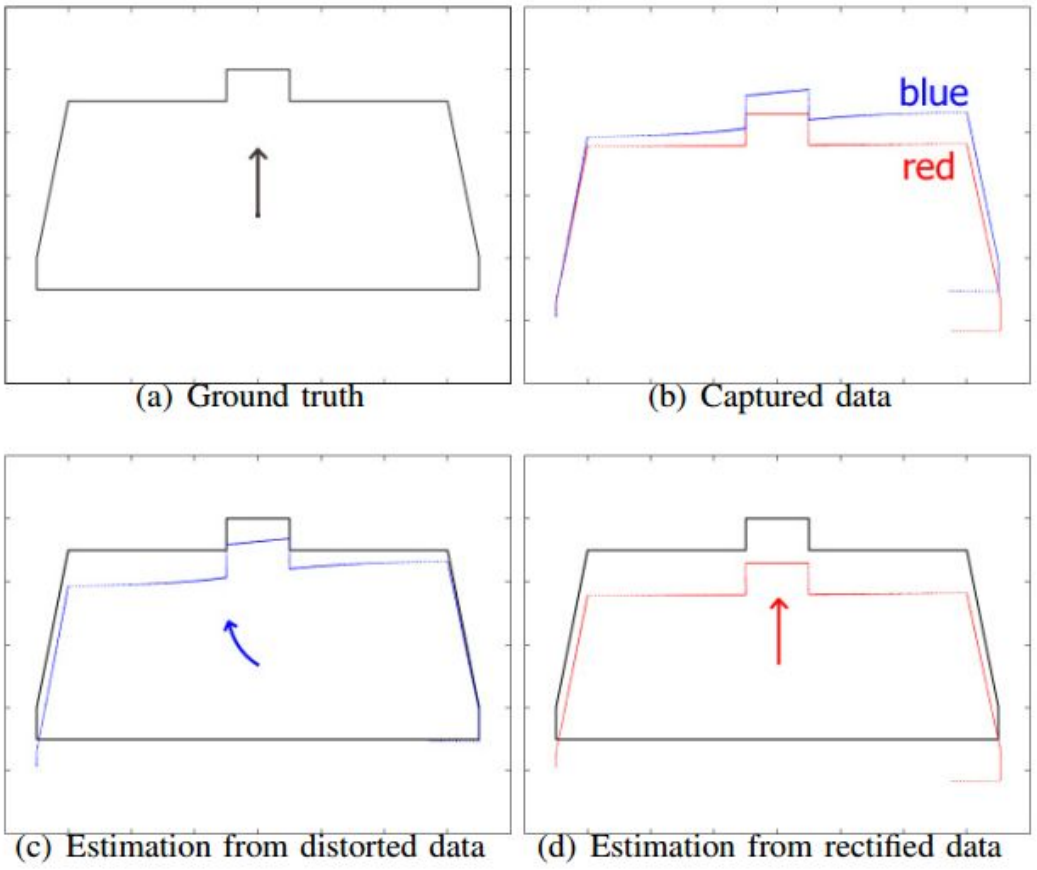


Figure 2, Laser scan distortion under motion leads to a wrong transformation estimation

First, we explain the details of the scan. X^i is a scan at time t^i. Similarly, X^{i-1} is a scan at time t^{i-1}. A time difference between two scans, X^i and X^{i-1}, is Δt. Each of the scans has its own local coordinate frame and a transformation T^i represents a coordinate frame of X^i. Therefore, the relative transformation from the coordinate frame of X^i to the coordinate frame of X^{i-1} can be given as $T_{i-1}^{-1}T_i$. Then a relation between X^i and X^{i-1} is represented as $x_k^{i-1} = T_{i-1}^{-1}T_i x_k^i, k = 1, ..., n$.

In order to compensate for the scan, the velocity of the rangefinder has to be estimated. We assume that the velocity is constant during the scanning time. Let V_i represent a body velocity of the coordinate frame of the rangefinder at time t_i. First, V_i is approximated from the relative transformation between X^i and X^{i-1} using a backward difference: $V_i = \frac{1}{\Delta t}T_i^{-1}T_i \approx \frac{1}{\Delta t}logT_i^{-1}T_i$. Then the approximated V_i is used to transform further each point in X^i. Let n denotes the number of points in X^i. Then a time difference between adjacent points is $\Delta t_s = \Delta t/n$ as shown in Figure 3.

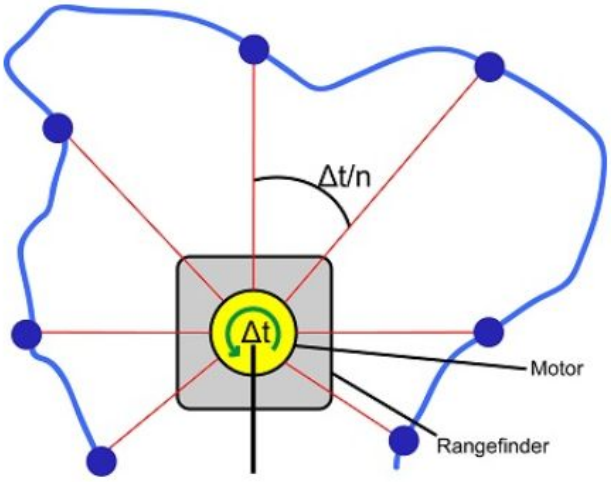


Figure 3, time difference between adjacent point in one scan

To be specific, $x_0, x_1, ..., x_n$ are points in X^i, and $t_{x_j} - t_{x_{j-1}}, (j = 0, 1, ..., n-1)$ equals to Δt_s. Each point in X^i has its own local coordinate frame and a transformation $T(t_i + \Delta t_s)$.

There is a trick that we use a backward compensation scheme to avoid a one cycle latency. By using the time when the last point is measured as a reference time(t_i), each of the points($x_0, x_1, ..., x_{n-1}, x_n$) has its own time instance($t_i - n\Delta t_s$, $t_i - (n-1)\Delta t_s, ..., t_i - \Delta t_s, t_i$). Therefore the transformation is $T(t_i - (n-j)\Delta t_s) = T_i e^{(n-j)\Delta t_s(-V_i)}$.
Plugging the transformation equation to the equation $x_k^{i-1} = T_{i-1}^{-1} T_i x_k^i, k = 1, ..., n$, X^i is converted into $\overline{X^i}$ which is compensated considering the robot motion: $\overline{X^i} = \left\{ e^{(n-j)\Delta t_s(-V_i)} x_j \mid j = 0, ..., n \right\}$.
After that, the original X^i compensated as $\overline{X^i}$, and the motion distortion are removed(Hong et al., 2010).

3.2 Probabilistic Motion Model

Robot kinematics has been studied thoroughly in past decades, however, it has almost exclusively been addressed in deterministic form. Probabilistic robotics generalizes kinematic equations that should be described in probabilistic motion models, because the outcome of control is uncertain, due to control noise or unmodeled exogenous effects.

For sampling models like GMapping, the algorithm for probabilistic motion model should be sampling from $p(x_t | u_t, x_{t-1})$, rather than a closed-form expression. The algorithm sample_motion_model_odometry, shown in Figure 4, implements the sampling approach, will be applied in this thesis. It accepts an initial pose x_{t-1} and an odometer reading u_t as input, and outputs a random x_t distributed according to $p(x_t | u_t, x_{t-1})$. Here the pose at time t is represented by $x_{t-1} = (x\ y\ \theta)^T$. The control is a differentiable set of two pose estimates obtained by the robot's odometer: $u_t = (\overline{x_{t-1}}\ \overline{x_t})^T$, with

$\overline{x_{t-1}} = \left(\bar{x}\ \bar{y}\ \bar{\theta}\right)^T$ and $\overline{x_t} = \left(\bar{x}\bar{y}'\bar{\theta}'\right)^T$. This model randomly guesses a pose x_t (lines 5-10), it is easy to implement(Thrun et al., 2002).

1: **Algorithm sample_motion_model_odometry(u_t, x_{t-1}):**

2: $\delta_{\text{rot1}} = \text{atan2}(\bar{y}' - \bar{y}, \bar{x}' - \bar{x}) - \bar{\theta}$

3: $\delta_{\text{trans}} = \sqrt{(\bar{x} - \bar{x}')^2 + (\bar{y} - \bar{y}')^2}$

4: $\delta_{\text{rot2}} = \bar{\theta}' - \bar{\theta} - \delta_{\text{rot1}}$

5: $\hat{\delta}_{\text{rot1}} = \delta_{\text{rot1}} - \mathbf{sample}(\alpha_1\delta^2_{\text{rot1}} + \alpha_2\delta^2_{\text{trans}})$

6: $\hat{\delta}_{\text{trans}} = \delta_{\text{trans}} - \mathbf{sample}(\alpha_3\delta^2_{\text{trans}} + \alpha_4\ \delta^2_{\text{rot1}} + \alpha_4\ \delta^2_{\text{rot2}})$

7: $\hat{\delta}_{\text{rot2}} = \delta_{\text{rot2}} - \mathbf{sample}(\alpha_1\delta^2_{\text{rot2}} + \alpha_2\delta^2_{\text{trans}})$

8: $x' = x + \hat{\delta}_{\text{trans}}\ \cos(\theta + \hat{\delta}_{\text{rot1}})$

9: $y' = y + \hat{\delta}_{\text{trans}}\ \sin(\theta + \hat{\delta}_{\text{rot1}})$

10: $\theta' = \theta + \hat{\delta}_{\text{rot1}} + \hat{\delta}_{\text{rot2}}$

11: *return* $x_t = (x', y', \theta')^T$

Figure 4, Algorithm for sampling from $p(x_t \mid u_t, x_{t-1})$ based on odometry information.

3.3 Mapping

In this project, we first study the mapping problem under the restrictive assumption that the robot poses are known. Put differently, we side-step the hardness of the SLAM problem by assuming some oracle informs us of the exact robot path during mapping. This problem is also known as mapping with known poses, in this project, two different methods were applied for comparison, they are occupancy grid mapping and TSDF mapping.

3.3.1 Occupancy grid mapping:

Occupancy grid mapping addresses the problem of generating consistent maps from noisy and uncertain measurement data. The basic idea of the occupancy grids is to represent the map as a field of random variables, arranged in an evenly spaced grid. Typically the grid cells can have three values, occupied, free or unknown. Probabilities can also be incorporated by giving each cell a probability of it being occupied which is then updated when new measurements are taken. Occupancy grid mapping algorithms implement approximate posterior estimation for those random variables.

Let m_i denote the grid cell with index i, an occupancy grid map m partitions space into finitely many grid cells: $m = \{m_i\}$. Each m_i has attached to it a binary occupancy value, which specifies whether a cell is occupied or

free. We will write “1” for occupied and “0” for free. The notation $p(m_i)$ refers to the probability that a grid cell is occupied.

The standard occupancy grid approach breaks down the problem of estimating the map into a collection of separate problems, namely that of estimating $p(m|z_{1:t}, x_{1:t}) = \prod_i p(m_i|z_{1:t}, x_{1:t})$ for all grid cell m_i.

Thanks to our factorization, the estimation of the occupancy probability for each grid cell is now a binary estimation problem with the static state. A filter for this problem was the binary Bayes filter, in this filter, the occupancy grid mapping algorithm uses the log-odds representation of occupancy: $l_{t,i} = log \frac{p(m_i|z_{1:t}, x_{1:t})}{1-p(m_i|z_{1:t}, x_{1:t})}$.

The advantage of the log odds over the probability representation is that we can avoid numerical instabilities for probabilities near zero or one. The probabilities are easily recovered from the log odds ratio: $p(m_i|z_{1:t}, x_{1:t}) = 1 - \frac{1}{1+exp\{l_{t,i}\}}$.

In the original occupancy grid algorithm, the inverse-sensor-model is used to update the occupancy value, unfortunately, fusing data from multiple sensors with Bayes filters is not an easy endeavor. However, robots are often equipped with more than one type of sensor. Hence, this question as to how to best integrate data from multiple sensors is particularly interesting if the sensors have different characteristics.

This question can be answered by occupancy grid mapping with forward models, it outputs the mode of the posterior and the mode is defined as the maximum of the logarithm of the map posterior: $m^* = argmax\ log\ p(m|z_{1:t}, x_{1:t})$

The map posterior factors into a map prior and a measurement likelihood: $log\ p(m|z_{1:t}, x_{1:t}) = const. + log\ p(z_{1:t}|x_{1:t}, m) + log\ p(m)$. Here $p(m)$ is the prior probability of occupancy, the log version of the prior is

$$log\ p(m) = const. + \sum_i m_i log\ p(m) - m_i log(1 - p(m)) = const. + \sum_i m_i l_0$$

The constant l_0 is dependent on the characteristics of sensors, therefore, it enables integrating information from more than one sensor into a single map, just by adding a correspondent constant l_0.

A hill-climbing algorithm for maximizing this log-probability is provided in Figure 5. This algorithm starts with an all-free map (line 2). It “flips” the occupancy value of a grid cell when such a flip increases the likelihood of the data (lines 4-6).

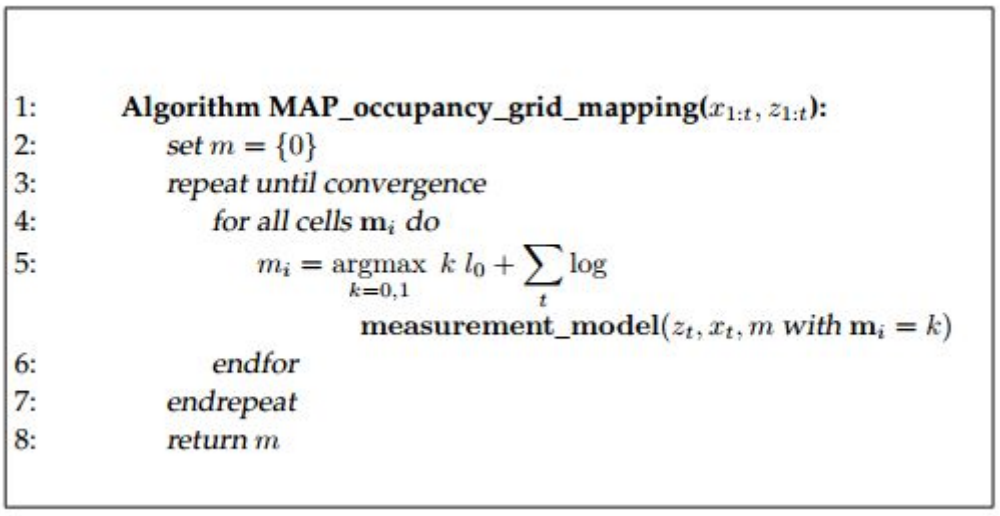


Figure 5, The maximum a posteriori occupancy grid algorithm

3.3.2 TSDF mapping:

TSDF is the abbreviation of Truncated Signed Distance Function, it is a volumetric representation of a scene for integrating depth images that has several benefits, e.g. time and space efficiency, representation of uncertainty or incremental updating.

The signed distance function (SDF) was proposed to reconstruct a 3D model from multiple range images. The environment is represented in a d-dimensional grid of equally sized voxels. The position of a voxel x is defined by its center. For each voxel, there are two relevant values. First, $sdf_i(x)$ which is the signed distance between voxel center and the nearest object surface in direction of the current measurement. In front of an object (in free space) the values are defined to be positive. Behind the surface (inside the object) distances are negative. Second, there is a weight $w_i(x)$ for each voxel to assess the uncertainty of the corresponding $sdf_i(x)$. The subscript i denotes the i'th observation. Figure 6 and the equation $sdf_i(x) = depth_i(pic(x)) - cam_z(x)$ define $sdf_i(x)$ precisely.

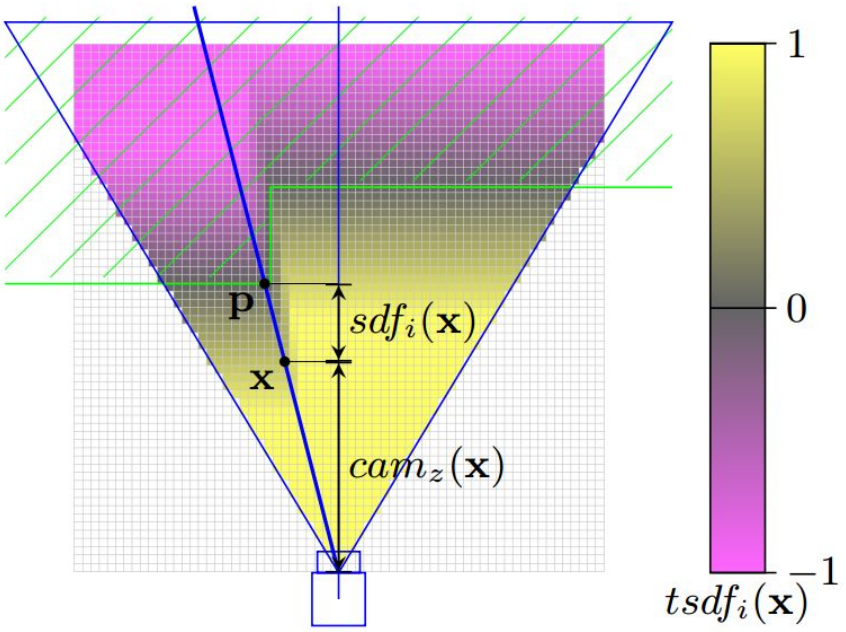


Figure 6, 2D TSDF. Solid object (green), camera with field of view, optical axis and ray (blue), and TSDF grid (unseen voxels are white, for others see color bar). The signed distance value of voxel x is determined by the depth of the corresponding surface point p and the voxel's camera distance $cam_z(x)$.

The $pic(x)$ is the projection of the voxel center x onto the depth image. So $depth_i(pic(x))$ is the measured depth in between the camera and the nearest object surface point p on the viewing ray crossing x. Accordingly, $cam_z(x)$ is the distance in between the voxel and the camera along the optical axis. Consequently, $sdf_i(x)$ is a distance along the optical axis as well. As to TSDF, it is the SDF has been truncated at $\pm t$. This is beneficial, because large distances are not relevant for surface reconstruction and a restriction of the value range can be utilized to memory footprint. The truncated variant of $sdf_i(x)$ is denoted by $tsdf_i(x): tsdf_i(x) = max(-1, min(1, \frac{tsdf_i(x)}{t}))$.

As mentioned above, multiple observations can be combined in one TSDF to integrate information from different viewpoints to improve accuracy or to add missing patches of the surface. This is done by weighted summation, usually through iterative updates of the TSDF. $TSDF_i(x)$ denotes the integration of all observations $tsdf_j(x)$ with $1 \le j \le i$. $W_i(x)$ assesses the uncertainty of $TSDF_i(x)$. A new observation is integrated by applying the following update step for all voxels x in the grid:

$$TSDF_i(x) = \frac{W_{i-1}(x)TSDF_{i-1}(x)+w_i(x)tsdf_i(x)}{W_{i-1}(x)+w_i(x)}$$

$$W_i(x) = W_{i-1}(x) + w_i(x)$$

In above function, the grid is initialized with $TSDF_0(x) = 0$ and $W_0(x) = 0$.

Most approaches set the uncertainty weight to $w_i(x) = 1$ for all updated voxels and to $w_i(x) = 0$ for all voxels outside the camera's field of view. This simply averages the measured TSDF observations over time.

For surface reconstruction, one can think of the TSDF like a level set. To find the object surface you look for the zero levels. This is usually done by ray casting from a given camera viewpoint. For each considered ray the TSDF is read step by step until there is a switch in sign. The information of surrounding TSDF values is then interpolated to estimate the refined point of zero crossings along the ray. This point is returned as an object surface point, like the boundary shows in Figure 7.

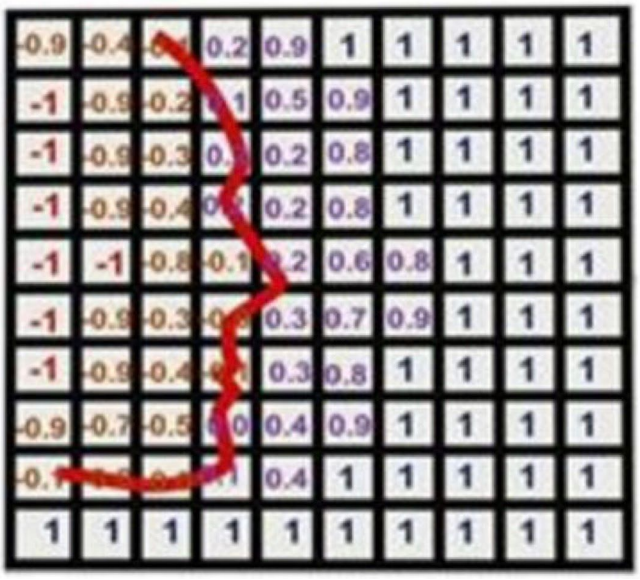


Figure 7, TSDF field and the object surface

4. **Experimental Setup**

ROS framework is used for data processing in this thesis which constrained all the algorithms evaluated on the same dataset which is available as ROS packages.

In addition to the communication framework, ROS and the accompanying community provide numerous tools and libraries which enable developers to concentrate on their main problem instead of developing supporting software. Commonly used tools include rviz visualization tool and rosbag recording/playback tool. Rviz enables easy visualization of many message types from point clouds to trajectories. The transformation library tf is an important part of the ROS framework as mobile robots commonly have various coordinate frames(for sensors, tools and the robot itself) requiring data to be transformed between them. The tf library listens to messages containing transformations between different coordinate frames from all nodes and forms a tree structure of the frames for easy traversal(Kaijaluoto et al., 2015).

4.1 preprocessing

Once a message named “Lidar” is received in ROS, it can be decoded as a timestamp and the raw data of one scan of a range finder. As IMU has a higher sampling frequency, we can get the corresponding IMU value at the given timestamp by linear interpolation. At the start of this scan, get the robot pose $frame\_start\_pose$ from IMU, and the $frame\_end\_pose$ pose at the end of this scan, the $frame\_end\_pose - frame\_start\_pose$ is the motion during this

scan, it will be uniformly distributed on 720 points of this scan and compensate on their location. The truly robot pose is the $frame\_base\_pose$, it is from the highest scoring particle, not from IMU, because IMU has “drift” after long time running, but the $frame\_end\_pose - frame\_start\_pose$ is accurate.

There are some transformation and rotation in distortion compensation, the coordinate transformation code as follows:

```
Eigen::MatrixXd Tob=Eigen::MatrixXd(3,3);
Tob<<cos(lastYaw),-sin(lastYaw),lastX,
        sin(lastYaw), cos(lastYaw),lastY,
        0       ,       0       ,  1;
Eigen::MatrixXd Toa=Eigen::MatrixXd(3,3);
Toa<<cos(currentYaw),-sin(currentYaw),currentX,
        sin(currentYaw), cos(currentYaw),currentY,
                0       ,       0       ,  1;
Eigen::MatrixXd Tba=Tob.inverse()*Toa;
pointAfterCalc=Tba*pointBeforeCalc;
```

4.2 motion model

Applying the sample_motion_model based on the odometry algorithm, in our project, the odometry is IMU. As we know, IMU suffers from accumulated error which may lead to drift over a long time, but it is precise enough in a short time like the interval between two frames. The algorithm in Figure 4 can be coded as follows:

```
double alpha1=atan2(delta.y, delta.x)-pold.theta;
double alpha2=delta.theta-alpha1;
noisypoint.theta+=sampleGaussian(stt*fabs(alpha1)+srt*sqrt(delta.x*delta.x+d
elta.y*delta.y));
noisypoint.x+=sampleGaussian(srr*fabs(delta.x)+str*fabs(delta.theta)+sxy*fab
s(delta.y));
noisypoint.y+=sampleGaussian(srr*fabs(delta.y)+str*fabs(delta.theta)+sxy*fab
s(delta.x));
noisypoint.theta+=sampleGaussian(stt*fabs(alpha2)+srt*sqrt(delta.x*delta.x+d
elta.y*delta.y));
```

4.3 mapping

In this process, taking one scan for explanation. In this software, “line” is the grid where this scan passes by, starts from the sensor center and ends at the “p_end” where the object is located.

4.3.1 Occupancy grid mapping:

In this method, all points on the line should be updated, except the end point “p_end”, other points should be updated by “miss”.
The core code for this method as follows:

```
for (int i=0; i<line.num_points-1; i++)
{
        map.cell(line.points[i]).pMapMisses++;;
}
map.cell(p_end).pMapHits++;
```

4.3.2 TSDF mapping:

In TSDF method, just compute the surface point, hence only the end point(where laser hit an object) and it's neighbour will be updated, but in different weight, the code as follows:

```
#define NEIGHBORDIST 1
#define LOGFREE -1
#define LOGOCC 2
for (int i=0; i<line.num_points-1; i++)
{
        if(abs(line.points[i].x-p_end.x)<=NEIGHBORDIST &&
abs(line.points[i].y-p_end.y)<=NEIGHBORDIST)
        {
                pMapTSDF=(visits*pMapTSDF)/(visits+1);
                visits++;
        }
}
pMapTSDF=(visits*pMapTSDF+1)/(visits+1);
visits++;
```

In this code, the smaller the NEIGHBORDIST value, the thinner the wall lines, which means the map has more certainty.

5. Results

5.1 sensor data preprocessing

In this part, we use another dataset for a better demonstration of the motion distortion correction, because in this dataset, the robot moved back and forth in a long corridor at high speed, it shows better the effect of our algorithm. As we know, the faster the robot moves, the more severe the motion distortion.

In Figure 8, the red points are raw sensor data which demonstrate the two walls in the corridor, but it shows different position and different orientation in forward and backward moving. This problem is caused by Laser scan distortion under motion. After removing the distortion as we introduced above, rectified datas are obtained and shows the walls as the green lines.

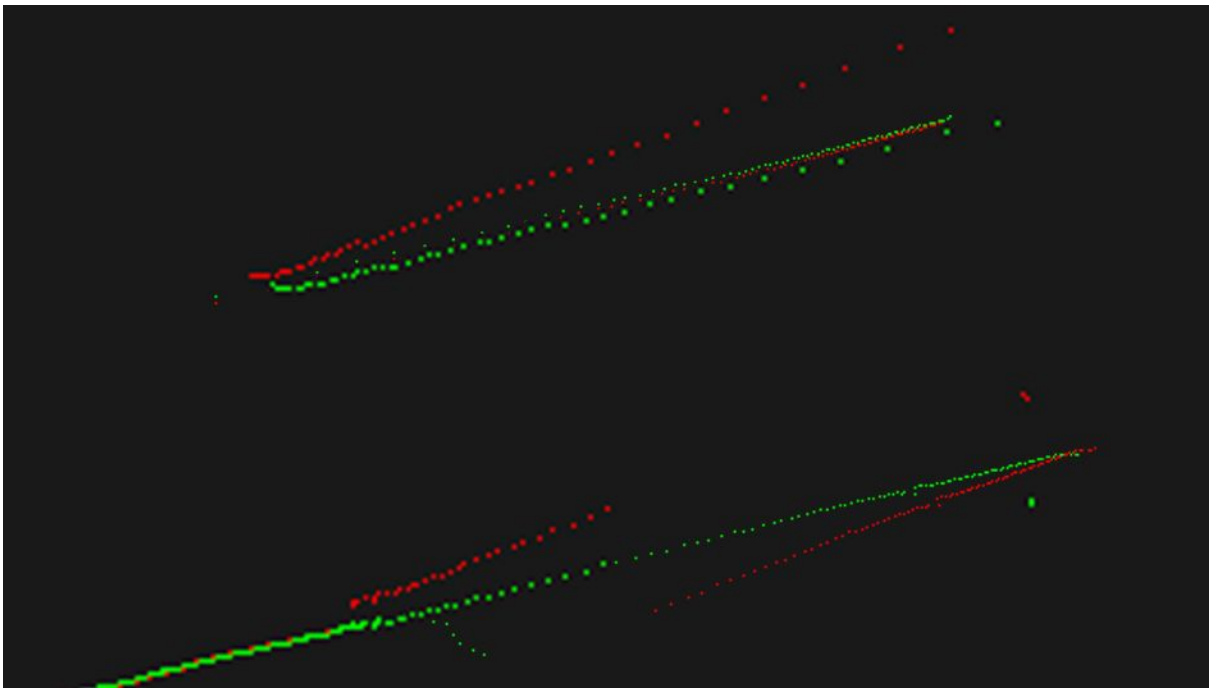


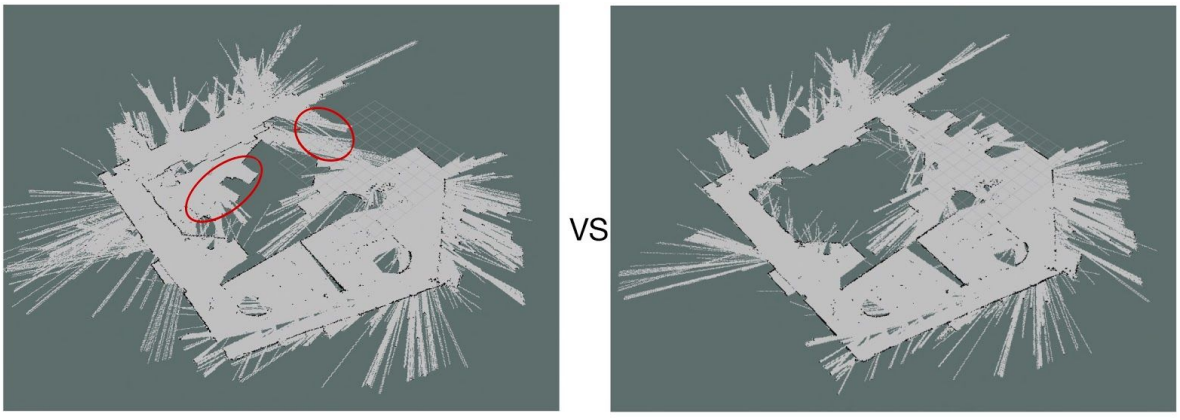
Figure 8, robot moves at a corridor

Before we add the distortion correction in sensor data preprocessing, the original project assumes that all points in one scan are measured simultaneously, it is prone to be erroneous under a fast motion. With our modification, the distortion caused by motion is removed, it increases the accuracy in our SLAM system.

This function is not necessary in all SLAM systems, it depends on the speed of the robot and the sensor. When the sensor scan rate is high compared to its extrinsic motion, motion distortion within the scan can often be neglected(Zhang et al., 2014).

5.2 motion model

There is an optimization on the motion model, the improvement can be shown in the map. In Figure 9(a), there are some errors. With the improvement of the sample motion model, this error does not exist in Figure 9(b).



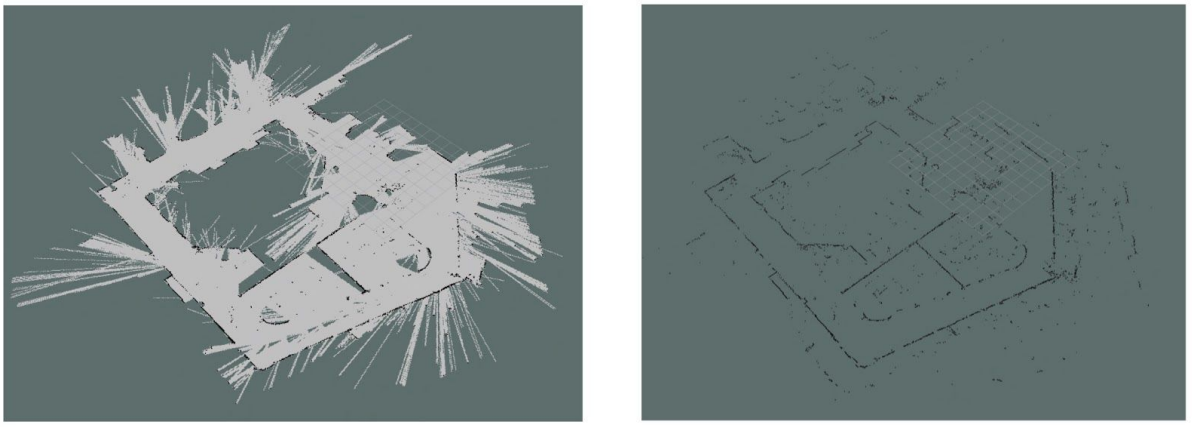
(a) Original motion model (b)Sample motion model based on odometry

Figure 9, the maps generated by two different motion model

5.3 mapping

In the mapping process, two methods are applied and the result shows as Figure 10. The Figure 10(a) is an occupancy grid map, it contains information not only about objects in the environment, but also about the absence of objects(e.g., free-space). This kind of map is great for mobile robot navigation: They make it easy to find paths through the unoccupied space. However, such grid-based approaches are computationally expensive and typically require a huge amount of memory.

On the contrary, the TSDF map in Figure 10(b) only contains boundaries and obstacles of the scenario, it costs less memory, therefore, it is usually used in making ground truth maps. In the SLAM algorithm, generating a map with the same format with the ground truth map would be beneficial for matching.



(a) Occupancy grid map (b)TSDF map

Figure 10, maps generated by two new mapping methods

6. Conclusion

The experiment results show that the sensor data preprocessing and the sample-motion-model improved the performance significantly.

In addition, two new map generation algorithms were applied in the mapping process, when compared with the original mapping algorithm, my occupancy grid map has the same precision and accuracy as the original one, but more robust and faster in calculation. As to the TSDF map, it only contains only boundaries and obstacles of the scenario, the small data enables the SLAM algorithm applying in large open spaces, because this kind of map costs much less memory than occupancy grid map.

7. References

Grisetti, G., Stachniss, C., & Burgard, W. (2007). Improved techniques for grid mapping with rao-blackwellized particle filters. IEEE transactions on Robotics, 23(1), 34-46.

Thrun, S. (2002). Probabilistic robotics. Communications of the ACM, 45(3), 52-57.

Hong, S., Ko, H., & Kim, J. (2010, May). VICP: Velocity updating iterative closest point algorithm. In 2010 IEEE International Conference on Robotics and Automation (pp. 1893-1898). IEEE.
Kaijaluoto, R. (2015). Precise indoor localization for mobile laser scanner.
Filipenko, M., & Afanasyev, I. (2018, September). Comparison of various slam systems for mobile robot in an indoor environment. In 2018 International Conference on Intelligent Systems (IS) (pp. 400-407). IEEE.
Zhang, J., & Singh, S. (2014, July). LOAM: Lidar Odometry and Mapping in Real-time. In Robotics: Science and Systems (Vol. 2, No. 9).