Simultaneous localization and mapping (SLAM) and application for autonomous vehicle

Student:

Yutian ZHANG

Teacher:

Yassine RUICHEK

Zhi YAN

Summary

Introduction

“Simultaneous localization and mapping (SLAM) and application for autonomous vehicle” is about that it’s hoped that the robot will start from an unknown location in an unknow environment, locate its own position and posture repeatedly observing map features (such as corners, columns, etc.) during the course of the autonomous movement in a vehicle, and then builds the map incrementally according to its own position so as to achieve simultaneous positioning, with the way to achieve the purpose of map construction.

- Introduction of SLAM

  In robotic mapping and navigation, simultaneous localization and mapping (SLAM) which also known as CML (Concurrent Mapping and Localization) is the computational problem of constructing or updating a map of an unknown environment while simultaneously keeping track of an agent's location within it. For now, there are several algorithms known for solving it, at least approximately, in tractable time for certain environments. Popular approximate solution methods include the particle filter, extended Kalman filter, and GraphSLAM.

  Considering the part of operation, there are two main problem in this topic, mapping: how the environment looks like and simultaneous localization: how the robot locates real-time itself. The core part of map construction is the expression of the environment and the interpretation of sensor data, and robot by recording information acquired in some form of perception, compares with current perception results to support the evaluation of the positioning of reality. And under the error and noise conditions, the complexity of positioning and map construction does not support the simultaneous solution of both, so it should bind both processes in a loop to support both parties to obtain a continuous solution in their respective processes, make the mutual iterative feedback in different processes in order to improve the continuous solution of both parties.

  And for the part of technology, as mentioned above, it’s about iterative mathematically to solve the problem of error and noise of environment and errors in iterations which all those errors will cumulative overlay, so the techniques of image matching methods or loop closure detection

methods and algorithms of Kalman filtering, particle filtering and scan matching data ranges were introduced.

- SLAM research status

  In the domain of SLAM, there are two main types: Visual SLAM whose sensor are cameras and Lidar SLAM whose sensor are lidar transmitter and receivers.

  SLAM systems become consumer products: Google Project Tango, Dyson 360 Eye among others, Autonomous Vehicles, Microsoft Hololens project etc. Some of the most important SLAM groups are now turning their attention to other problem domains: for instance, Prof. Andrew Davison at Imperial College London who now works more on scene understanding and robotic manipulation problems and Prof. Wolfram Burgard at University of Freiburg who is working on all sorts of miscellaneous problems. Some other leading labs of yester-years like those at University of Zaragoza seem to continue to work on SLAM, but the work is more focused on building systems like ORB-SLAM that integrate already existing ideas.

  With different usage of SLAM algorithms and SLAM types, that are tailored to the available resources, hence not aimed at perfection, but at operational compliance. Published approaches are employed in self-driving cars, unmanned aerial vehicles, autonomous underwater vehicles, planetary rovers, newer domestic robots and even inside the human body.

  The future development trend of SLAM has two major categories: First, it is toward lightweight and miniaturization, allowing SLAM to operate well on small devices such as embedded devices or mobile phones, and then considering it as an application of the underlying functions. After all, in most occasions, our real goal is to realize the functions of robots, AR/VR devices, such as sports, navigation, teaching, and entertainment, and SLAM is to provide its own position estimate for upper-level applications. In these applications, we do not want SLAM to occupy all computing resources, so there is a strong demand for SLAM's miniaturization and lightweight. On the other hand, high-performance computing equipment is used to achieve precise 3D reconstruction and scene understanding. In these applications, our goal is to reconstruct the scene perfectly, and there is no limit to the portability of computing resources and devices.

Projects choices

From all of the existing projects on the website of OpenSLAM, I choice two projects to carry out my projects: Gmapping in Lidar SLAM and RBGD-SLAM in visual SLAM.

- Gmapping

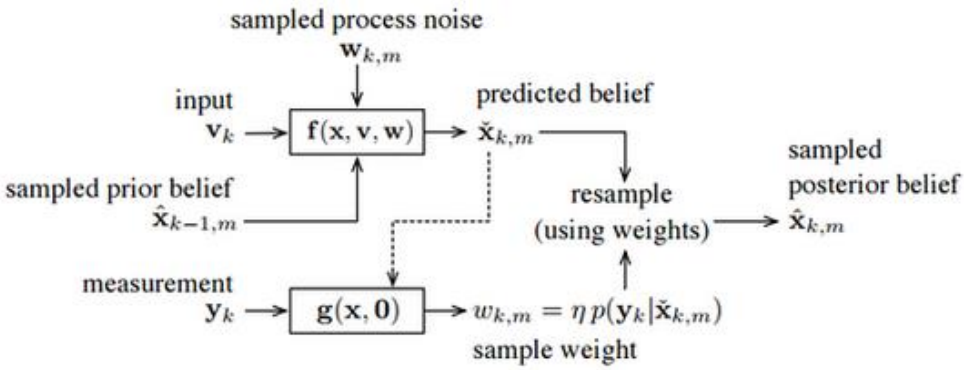
  Gmapping is a 2D Lidar SLAM with the algorithm RBPF (Rao-Blackwellized particle filter) which is highly to learn grid maps from laser range data, the algorithm of scan-match to locate the position of robot and the method of gradient descent.

  Gmapping uses RBPF as its algorithm, as a kind of PF (Particle Filter) algorithm, comparing to the other algorithms used in SLAM projects, like KF (Kalman Filter), EKF (Extended Kalman Filter), UKF (unscented Kalman Filter), IEKF (Iterated Extended Kalman Filter) etc. PF has a high accuracy in a low dimensional situation.

  And following are the comparison between these algorithm:

  1. PF (Particle Filter)

     PF uses the method of describing the distribution with a large number of sampling points, it uses Sequential Monte Carlo to collect sampling points every moment, it uses the loop of sampling—calculate weights—re-sampling:



The more particles that match the observation, the more weights they have, and the larger the weight, the easier it will be sampled.

PF updates the probability of each particle by comparing the difference between the observation value of each particle and the predicted value of the model.

Motion equation: $f_k : R^{n_x} \times R^{n_v} \to R^{n_x}$

Observation equation: $h_k : R^{n_x} \times R^{n_n} \to R^{n_y}$

n_x: status, n_v: dimension of noise vector during movement,
n_y: observation value, n_n: dimension of noise vector during observation

With Bayes' theorem, to build credibility density function equation and initial. The probability density function can be recursively calculated using the two steps of "prediction" and "update".

Prediction: PF considers the probability density function at *k-1* that it's known:

$$\begin{aligned} p(x_k \mid y_{1:k-1}) &= \int p(x_k, x_{k-1} \mid y_{1:k-1}) dx_{k-1} \\ &= \int p(x_k \mid x_{k-1}, y_{1:k-1}) p(x_{k-1} \mid y_{1:k-1}) dx_{k-1} \\ &= \int p(x_k \mid x_{k-1}) p(x_{k-1} \mid y_{1:k-1}) dx_{k-1} \end{aligned}$$

The status transition model is assumed to be a first-order Markov process, and the status of time *k-1* is determined only by minute *k*. $p(x_k \mid x_{k-1}, y_{1:k-1}) = p(x_k \mid x_{k-1})$,

Update: for observation value: y_k, PF uses previous posteriori probability to calculate the next posteriori probability.

$$p(x_k \mid y_{1:k}) = \frac{p(y_k \mid x_k) p(x_k \mid y_{1:k-1})}{p(y_k \mid y_{1:k-1})}$$

->

$$p(y_k \mid y_{1:k-1}) = \int p(y_k \mid x_k) p(x_k \mid y_{1:k-1}) dx_k$$

PF can get observation equation: $y_k = h_k(x_k, n_k)$

Sequential Importance Sampling: In each recursive process, it calculates the weight of the next sample from the weight of the previous sample, and approximate it with weighted average, it can get the weights equation: $w_k^{(i)} \propto \frac{p(x_k^{(i)} \mid y_{1:k})}{q(x_k^{(i)} \mid y_{1:k})}$ with *q(x)* important density. After breaking down important density as $q(x_k \mid y_{1:k}) = q(x_k \mid x_{k-1}, y_{1:k}) q(x_{k-1} \mid y_{1:k-1})$, then it can get the weights recursive expression

$$
\begin{aligned}
w_k^{(i)} &\propto \frac{p(y_k \mid x_k^{(i)}) p(x_k^{(i)} \mid x_{k-1}^{(i)}) p(x_{k-1}^{(i)} \mid y_{1:k-1})}{q(x_k^{(i)} \mid x_{k-1}^{(i)}, y_{1:k}) q(x_{k-1}^{(i)} \mid y_{1:k-1})} \\
&= w_{k-1}^{(i)} \frac{p(y_k \mid x_k^{(i)}) p(x_k^{(i)} \mid x_{k-1}^{(i)})}{q(x_k^{(i)} \mid x_{k-1}^{(i)}, y_{1:k})}
\end{aligned}
$$

.

Resampling: Sequential Importance Sampling will cause degeneracy problem that it will waste lots of calculation in useless particles because only a small number of particles have more weights after several recursions. So “Effective number of particles” is defined as $\widehat{N_{eff}} = \frac{1}{\sum_{i=1}^{N} (w_k^{(i)})^2}$, and we can set a threshold, when effective number of particles is less than this threshold, do resampling to avoid degeneracy problem.

2. KF (Kalman Filter)

KF uses Linear Gaussian system, in a Linear Gaussian system, the motion equations and observation equation are linear, and the two noise terms obey the zero-mean Gaussian distribution, after linear transformation, Gaussian distribution is still Gaussian distribution. With Bayes’ theorem, to calculate the posterior probability distribution of *x* and get the results. Algorithm of KF considers all the environment and situation as Linear Gaussian system. But the systems in reality are not linear, status and noise

are not distributed as Gaussian distribution, so it won't be very accurate about the results.

Status estimation of discrete-time systems:

$$\begin{cases} x_k = f\left(x_{k-1}, u_k, w_k\right) \\ y_k = g\left(x_k, n_k\right) \end{cases}$$

Among them, x_k: status at time *k*, f(x): motion equation, u: input, w: noise input, g(x): observation equation, y: observation data, n: observation noise.

With odometer and sensor (lidar sensor in Gmapping) in robot, here are the motion equation: $x_{k+1} = x_k + \Delta x_k + w_k$ and observation equation: $\begin{bmatrix} r \\ \theta \end{bmatrix}_k = \begin{bmatrix} \sqrt{\|x_k - L_k\|_2} \\ \tan^{-1} \frac{L_{k,y} - x_{k,y}}{L_{k,x} - x_{k,x}} \end{bmatrix} + n_k$.

Linear system:

$$\begin{cases} x_k = A_{k-1} x_{k-1} + u_k + w_k \\ y_k = C_k x_k + n_k \\ w_k \sim N\left(0, Q_k\right) \\ n_k \sim N(0, R_k) \end{cases}$$

Among them, Q_k, R_k are the covariance matrix of two noise, A, C are stochastic matrix and observation matrix.

With Bayes' theorem, to calculate the MAP (maximize a posteriori probability) estimate about *x*,

$$\begin{aligned} \hat{x} &= \arg\max_x p\left(x|y, v\right) \\ &= \arg\max \frac{p(y|x,v)p(x|v)}{p(y|v)} \\ &= \arg\max p(y|x)p\left(x|v\right) \end{aligned}$$

Finally MAP estimate is:

$$\hat{x} = \arg\min \sum_{k=0}^{K} J_{y,k} + J_{v,k}$$

by recursively solving this equation with matrix, here are the results:

$$\tilde{P}_k = A_{k-1}\hat{P}_{k-1}A_{k-1}^T + Q_k$$
$$\tilde{x}_k = A_{k-1}\hat{x}_{k-1} + v_k$$
$$K_k = \tilde{P}_k C_k^T \left(C_k \tilde{P}_k C_k^T + R_k\right)^{-1}$$
$$\hat{P}_k = \left(I - K_k C_k\right)\tilde{P}_k$$
$$\hat{x}_k = \tilde{x}_k + K_k\left(y_k - C_k\tilde{x}_k\right)$$

$\tilde{P}_k$ and $\tilde{x}_k$ are prediction equations by last moment, K_k is Kalman Gain, $\hat{P}_k$ and $\hat{x}_k$ are correction equation.

3. EKF (Extended Kalman Filter)

   KF considers that the system is NLNG (non-linear non-Gaussian), so after non-linear transformation, Gaussian distribution is no longer Gaussian distribution, EKF uses the way of approximation to approach the results, it uses Gaussian distribution to approximate this situation and linearizate the system around the operation points. But there are lots of problems that will make errors when approximation.

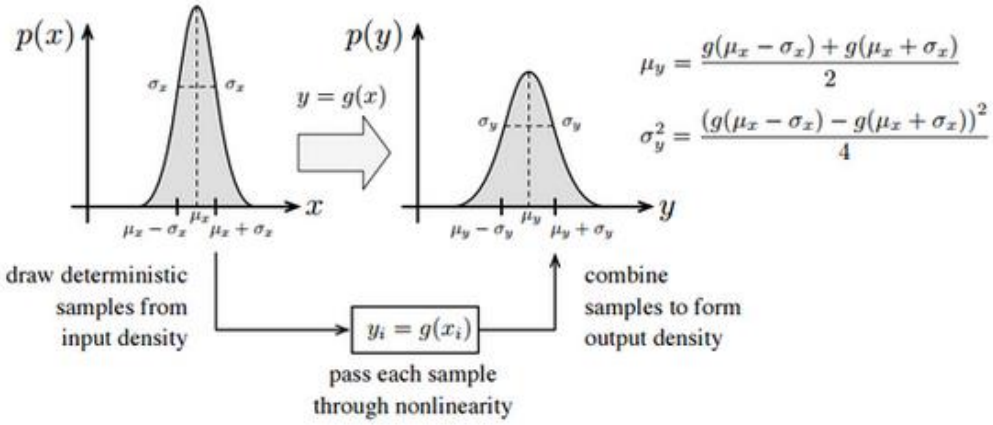
4. IEKF (Iterated EKF)

   In order to solve the one of the problem of EKF: the operation point is an estimated mean but not real mean of the input status by linearizing operation points. So IEKF approximate the estimated mean by calculating EKF again with the operation point that EKF offers, to get another operation point at the same moment, by iterating calculating until this operation point change is small enough.

5. UKF (unscented Kalman Filter)

   In order to solve the one of the problem of EKF: EKF estimates the mean of nonlinear output with linear system, the same as calculating covariance, so the expectations are not real. So, UKF approximates the situation NLNG with Gaussian distribution after choosing a point as operation point by distributing the sigma points. Because the operation points in EKF are not accurate, so

UKF choices some sample points as Sigma Points, with Gaussian distribution after mapping all the Sigma Points, by this way to approach the real results.



So, comparing all these algorithms, I think PF (Particle Filter) calculates the situation without adding parameters to calculate, so it can avoid a lot of errors during calculation. But there is a big problem: the number of particles required for sampling, exponentially increasing with the distribution. However, it will require a lot of calculation in describing the distribution. Then PF can only solve the situation in a low dimension environment.

But for RBPF, it introduces the adaptive techniques to reduce the number of particles in a Rao- Blackwellized particle filter for learning grid maps. We propose an approach to compute an accurate proposal distribution taking into account not only the movement of the robot but also the most recent observation. This drastically decrease the uncertainty about the robot's pose in the prediction step of the filter. Furthermore, we apply an approach to selectively carry out re-sampling operations which seriously reduces the problem of particle depletion.

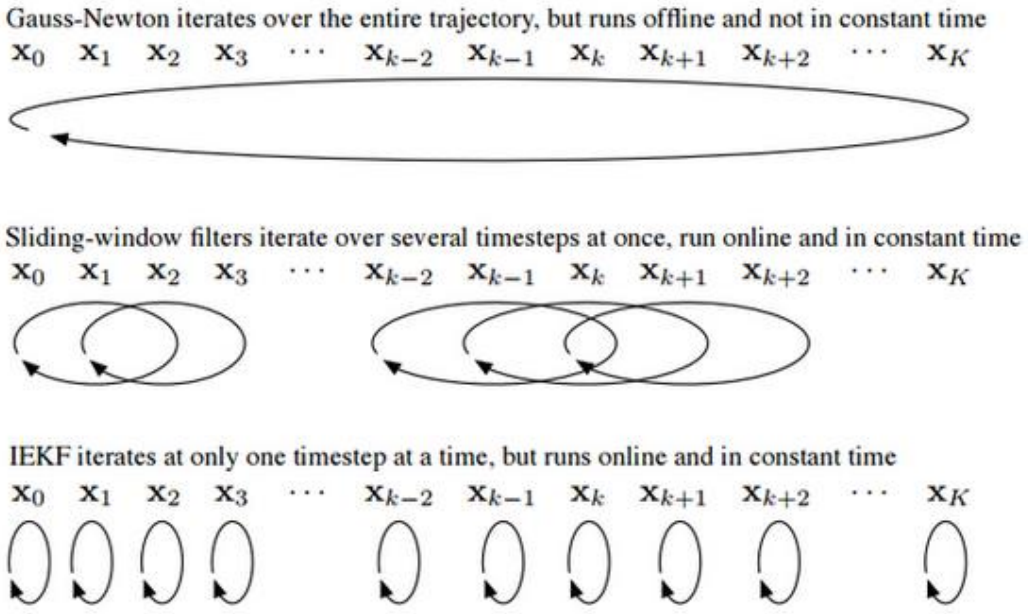
- RGBD-SLAM

RGBDSLAM allows to quickly acquire colored 3D models of objects and indoor scenes with a hand-held Kinect-style camera. It provides a SLAM front-end based on visual features s.a. SURF or SIFT to match pairs of acquired images and uses RANSAC to robustly estimate the 3D transformation between them. The resulting camera pose graph is then

optimized with the SLAM back-end HOG-Man. To achieve online processing, the current image is matched only versus a subset of the previous images. Subsequently, it constructs a graph whose nodes correspond to camera views and whose edges correspond to the estimated 3D transformations. The graph is then optimized with HOG-Man to reduce the accumulated pose errors.

1. Algorithm: Nonlinear Optimization

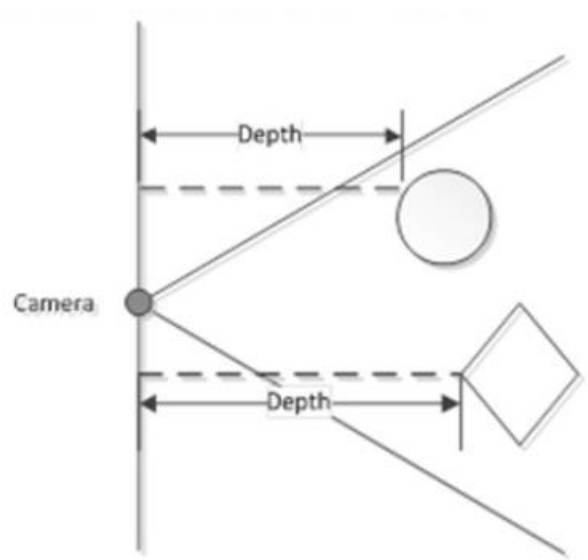
   As a visual SLAM project, it uses the algorithm of nonlinear optimization, it calculates MAP (maximize a posteriori probability) as well, it makes errors equation and adds it into motion equation and observation equation. By calculating, it just considers that noise is satisfied with Gaussian distribution, then optimize it by using gradient descent optimization algorithm to calculate the results from initial.



   Comparing to algorithm of filters, it can consider the constraints in entire trajectory at a time, its linearization also considers to the entire trajectory.

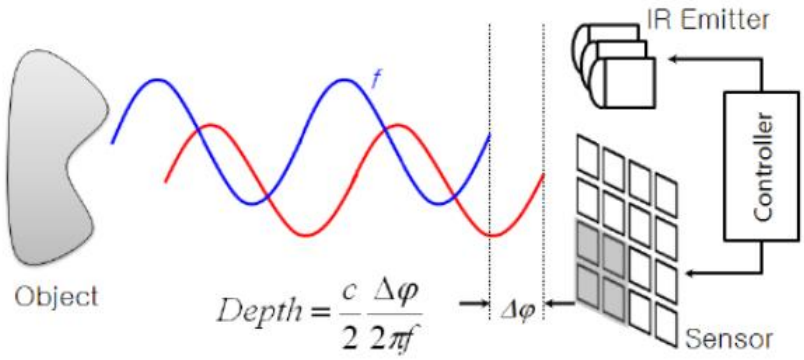
2. Front end: Depth camera

   RGBD uses Kinect v2 as the sensor of front-end, Kinect v2 carries a depth camera, one RGB color camera, one infrared transmitter and a depth sensor, RGB color camera is for collecting the images, and the combination of infrared transmitter and a depth sensor, is for collecting the depth (distance) between sensor and aims. This depth camera can collect depth maps that contains information relating to the distance of surfaces of scene objects from a viewpoint. Depth map can be calculated as point cloud data after coordinate conversion.



3. Front end: TOF (Time of Flight)

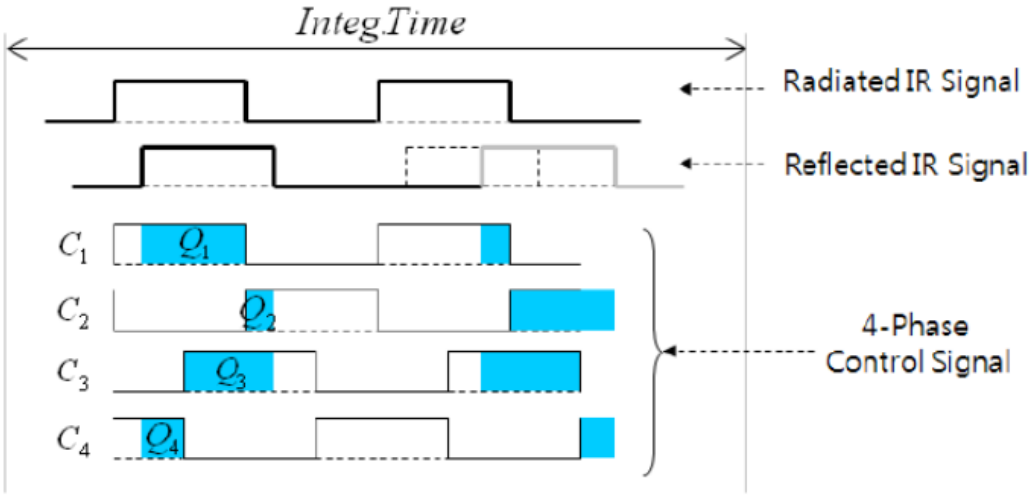
   Camera measure the phase-delay of transmitted and reflected IR signals, and then calculate the distance from each sensor pixel to the target object.



The delay phase difference t_d is calculated by the relationship of the four charge values. There is a 90-degree phase delay between these four phase control signals.

$$t_d = \arctan\left(\frac{Q_3 - Q_4}{Q_1 - Q_2}\right)$$

Q_1 to Q_4 represent the amount of electric charge for the control signals C_1 to C_4, the corresponding distance can then be calculated: $d = \frac{c}{2f}\frac{t_d}{2\pi}$, c: the speed of light, f: signal frequency.



The operating range is between 0.8 meters to 3.5 meters, the spatial resolution is 3mm at 2 meters distance, and the depth resolution is 10mm at 2 meters distance. The FOV is 57×43 (HxV) degrees.

It compares to other technologies, TOF is good at the performance of anti-interference and it offers wider viewing angle with middle accuracy.

4. Closed-loop detection

If the back-end like G2O, to do every calculation, the amount of data is too large to run fluently, so closed-loop detection can well reduce this problem, according to the appearance of a new image, see if it is similar to the previous key frame, it can also avoid the errors during the course of SLAM.

Project realization

- Gmapping

1. Simulation environment:

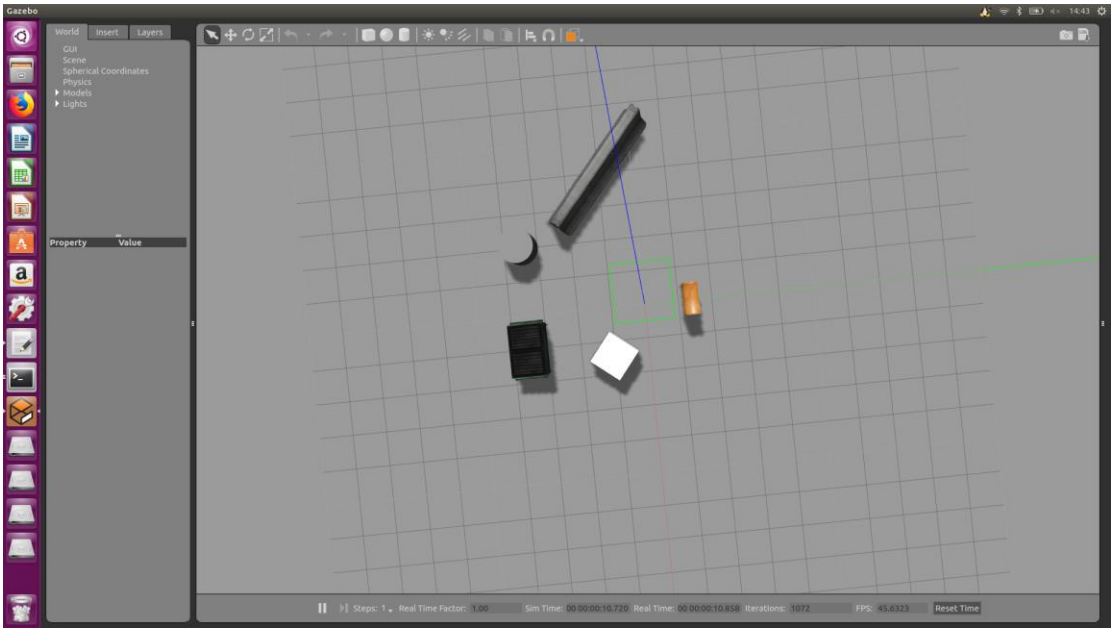
   Operation system: Ubuntu 16.04
   Ros version: Kinetic
   Gazebo version: 7.0
   Simulation Robot: Turtlebot

2. Preparation:

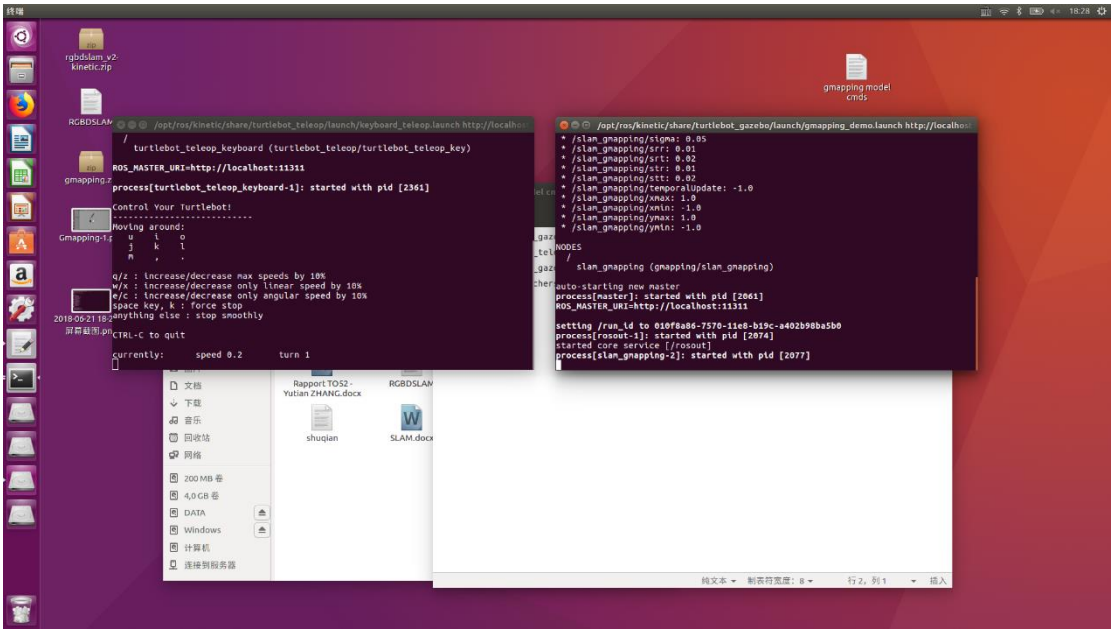
   Install the ROS full version kinetic, following the tutorial in website ROS.org, and ROS kinetic includes Gazebo 7.0 already;
   Install some components of ROS-Gazebo: "*ros-kinetic-gazebo-ros-pkgs ros-kinetic-gazebo-ros-control*";
   Install model package of Gazebo;
   Install related packages of turtlebot: "*ros-kinetic-turtlebot-**";

3. Simulation demo

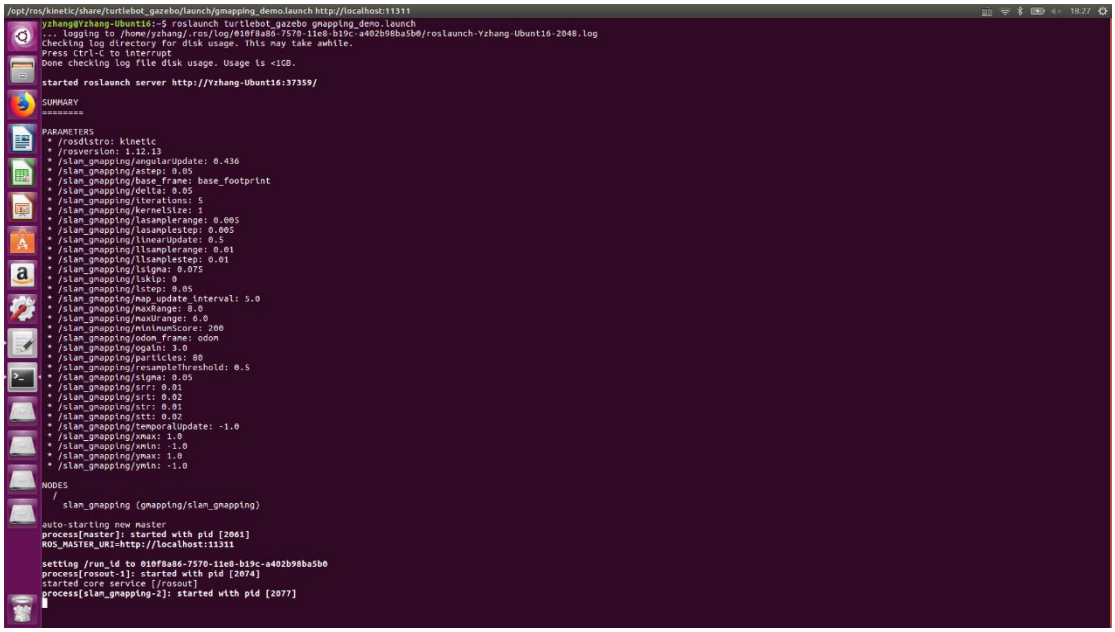
   1) Launch Gazebo, load turtlebot and environment model: "*roslaunch turtlebot_gazebo turtlebot_world.launch*";



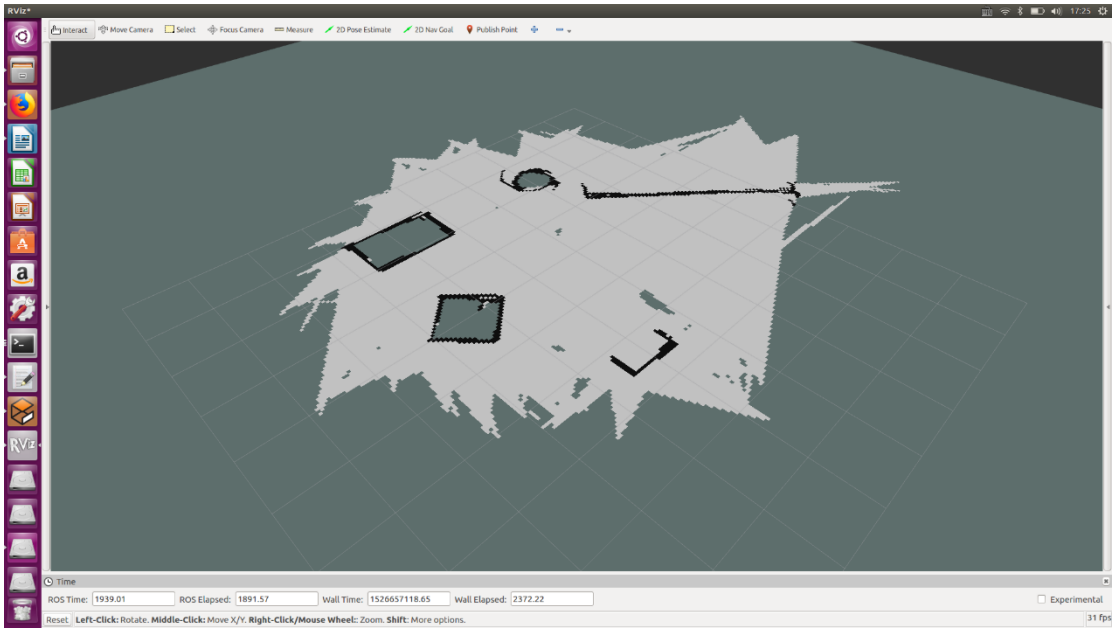
2) Launch keyboard control node to control turtlebot: "*roslaunch turtlebot_teleop keyboard_teleop.launch --screen*";



3) Launch Gmapping: "*roslaunch turtlebot_gazebo gmapping_demo.launch*";



4) Launch Rviz to observe the map building process: "*roslaunch turtlebot_rviz_launchers view_navigation.launch*";



- RGBDSLAM

1. Simulation environment:

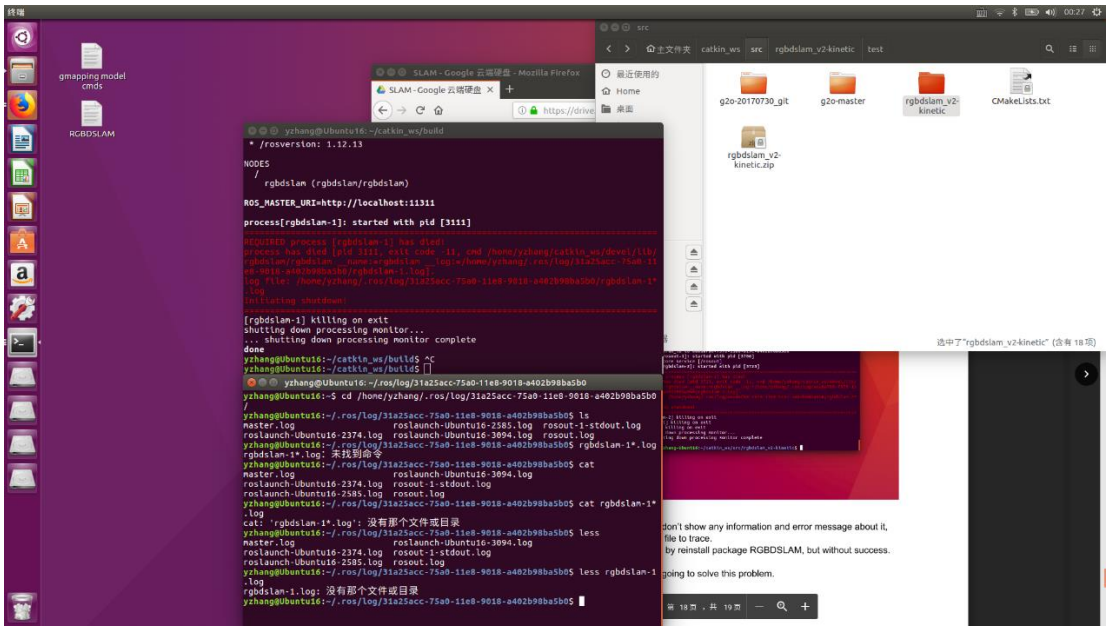
   Operation system: Ubuntu 16.04
   Ros version: Kinetic
   Other packages: G2O, RGBDSLAM package
   WorkStation: calkin_ws
   Dataset: rgbd_dataset freburg1_xyz.bag

2. Preparation:

   Install the ROS full version kinetic, following the tutorial in website ROS.org;
   Install RGBDSLAM in website ROS.org which relates to GitHub (met a problem about version of c++ standard, solved by adding *set(CMAKE_CXX_FLAGS "-std=c++11")* in file CMakeLists.txt)
   Install G2O in website ROS.org which relates to GitHub
   Install the independencies about G2O and RGBDSLAM
   Install a dataset: rgbd_dataset freburg1_xyz

3. Simulation demo

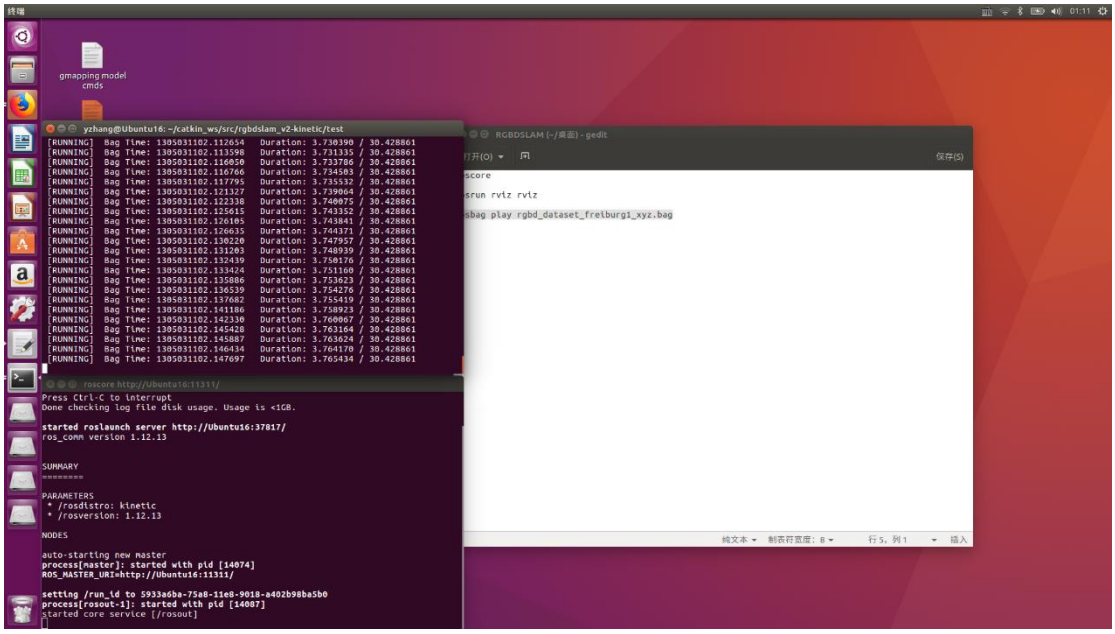
   When I want to launch rgbdslam with “*roslaunch rgbdslam rgbdslam.launch*”, there are the errors of following picture:

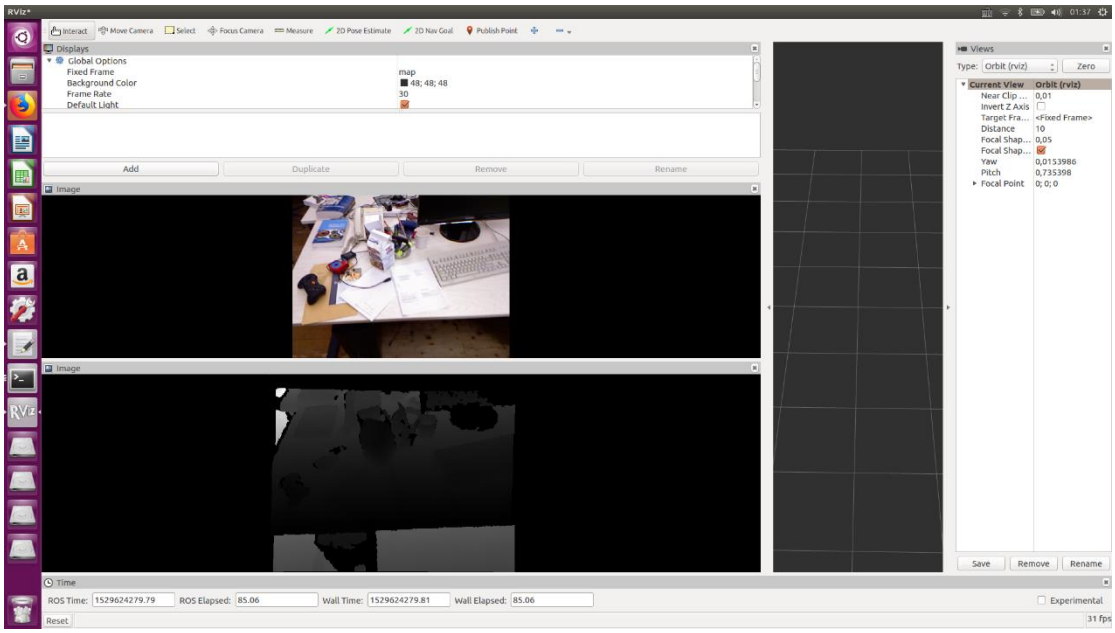


Errors don’t show any information and error message about it, not log file to trace.
Plan to by reinstall package RGBDSLAM, but without success. Still ongoing to solve this problem.

For depth map (depth image): with the dataset I download: rgbd_dataset freburg1_xyz.bag
roscore
rosbag play rgbd_dataset freburg1_xyz.bag

and open rviz: *rosrun rviz rviz*





Conclusion

During the period of this project, I do two sous-projects: Gmapping and RGBDSLAM that makes me learn SLAM step by step from knowing nothing to be familiar with SLAM. When I do these two projects, there are some problems all the time, some are easy to solve and some problems takes a lot of time doing research.

I think there is still a lot of space for development in the domain of SLAM about optimization of algorithms or materials both in front-end and back-end parts.

At last, I want to thanks the helps that mister RUICHEK and mister YAN give me to advance my project.

Annex

https://openslam-org.github.io/

https://en.wikipedia.org/wiki/Simultaneous_localization_and_mapping

https://www.cnblogs.com/gaoxiang12/p/5560360.html

https://en.wikipedia.org/wiki/Particle_filter

https://blog.csdn.net/qq_27550989/article/details/78341904

https://blog.csdn.net/bingoplus/article/details/56667475

https://blog.csdn.net/MyArrow/article/details/52678020

https://en.wikipedia.org/wiki/Depth_map

http://www.ros.org/

https://blog.csdn.net/lingchen2348/article/details/79503970

http://www.cnblogs.com/gaoxiang12/p/4462518.html