Proceedings of the 2017 IEEE
International Conference on Robotics and Biomimetics
December 5-8, 2017, Macau SAR, China

INDOOR MAPPING USING GMAPPING ON EMBEDDED SYSTEM

Qinjie Lin, Zhaowu Ke, Sheng Bi*,Sirui Xu, Yuhong Liang, Fating Hong, Liqian Feng

***Abstract*—In recent years, the core technology enable the rapid rise of autonomous vehicles like the solving the Simultaneous Localization And Mapping (SLAM) problem on the embedded system. And the gmapping package on the ROS platform providing laser-based SLAM is widely used and studied. However, implementing it on embedded system imposes two challenges to us: running ROS requires large CPU consumption and the algorithm of SLAM is computationally intensive. In this paper, we present a system implementation of the gmapping on an ARM based embedded hardware not installing ROS and we also improve the architecture of the mapping system by parallel execution of mapping and estimating pose. The mapping process is running on the embedded system and it can create a 2-D occupancy map from laser and pose data collected by a mobile robot with low CPU load and time consumption.Experimental results carried out with mobile robots in indoor environments illustrate the accuracy of our implementation and the low consumption of time and CPU load of the mapping system.**

Keywords— mapping; gmapping; mobile robot; SLAM;

I. INTRODUCTION

Recently the development of intelligent robots is so fast, thanks to the advancements in sensors and processors design. And more and more researchers focus on combining the sensor with embedded system to implement some intelligent function like mapping. Then some attempts for real-time implementations of SLAM algorithm on embedded system were performed. [13] has developed a system architecture based on the co-design of a hardware architecture, a feature detector, a SLAM algorithm and an optimization methodology and [14] has proposed an effcient implementation of the EKF-SLAM algorithm on a multi-core architecture. However,it is still not a easy task to run some mapping process on the embedded system on which there is limited amount of CPU consumption. Following the idea of combining sensors with embedded system to enable robot to be intelligent, we aim to run a mapping process via an embedded processor to build maps for robots in our works.

Building maps is one of most basic yet important task of mobile robots. Actually by employing sensor like laser sensor, the robot can build maps from the information gathered with the robot's sensor and the pose data collected by a mobile robot. With the maps, the robots can locate themselves and navigate in the practical environment. Then location and navigation is the key to intelligent mobile robots. Therefore many researchers focus on the mapping problems of mobile robots and this kind of problem is referred as SLAM problem. Then, many methods were proposed to develop different SLAM algorithms like FAST SLAM[7], GRAPH SLAM [11] and DP-SLAM[12] which aim to improve accuracy, robustness or processing time.

Nowadays, many SLAM functions have been implemented with the development of SLAM. And the gmapping Package on the ROS platform provides laser-based SLAM as a ROS node and it can build maps. This method of building maps is widely used and have shown remarkable success. However, the main problems of running a ROS node to build maps for robots in a embedded operating system is the large consumption of CPU. Therefore the major challenge of building maps in embedded operating system is reducing the required CPU load for running mapping process. Besides, the gmapping algorithm proposed in [1] and [2] is computationally intensive and mapping process always has real-time requirement. So it may be impractical to implement the simple mapping system on the embedded system. In this case, we need to improve the architecture of mapping system by paralleling execution of some steps of the algorithm.

Based on the fact that running ROS requires a lot of CPU consumption on the embedded system and mapping process need real-time requirement, we proposed to build maps for robots not using ROS and parallelize the steps of building maps and estimating pose. In this way it required less CPU consumption and time for running mapping process on embedded system. Then we do experiments in indoor environment to validate our approach. In our experiments, the 2-D maps generated by our mapping process is highly accurate in the indoor environment.

This paper is organized as follows. After describing the related hardware used in our experiments, we describe how gmapping algorithm can be used in general to solve the SLAM problem in Section III. We then provide implementation details of our mapping systems in Section IV and present an improving architecture of mapping system in section V. Then in Section VI, experiments carried out on real robots are presented. Finally, Section VII concludes our work and describes the future work.

II. RELATED HARDWARE

In this section, we will firstly introduce the ARM board that we use to implement our mapping system.The we will describe a Laser Scanner that provides scan data to mapping process and the robot odometric model that returns odometric information to mapping process.

A. NanoPi2

In our experiments, we implement our mapping system on a NanoPi2 which supports Ubuntu. The NanoPi2 is a high performance ARM Board and the core featured on NanoPi2 is the Samsung Cortex-A9 Quad Core S5P4418 as shown in

Qinjie Lin, Zhaowu Ke, Sheng Bi*,Sirui Xu, Yuhong Liang, Fating Hong, Liqian Feng are with the School of Computer Science&Engineering, South China University of Technology, GuangZhou, 510640. (corresponding author: Sheng Bi, e-mail: picy@scut.edu.cn)

978-1-5386-3741-8/17/$31.00 © 2017 IEEE

figure2. The SoC consists of four-core CPU and the four cores share a L2 cache. Note that the I/O subsystem is directly connected to the CPU. If data reaches, the CPU will then process the data through shared memory. Additionally, its adoption of the Raspberry Pi's GPIO pin header makes it compatible with both Raspberry Pi's external GPIO modules and Arduino's shield boards so we can exchange the data with the laser sensor and the robot. As shown in figure1, its PCB dimension is 75 x 40 mm.

a) NanoPi2 b) The system Ubuntu 14.04.5 embedded on NanoPi2

Figure 1. NanoPi2

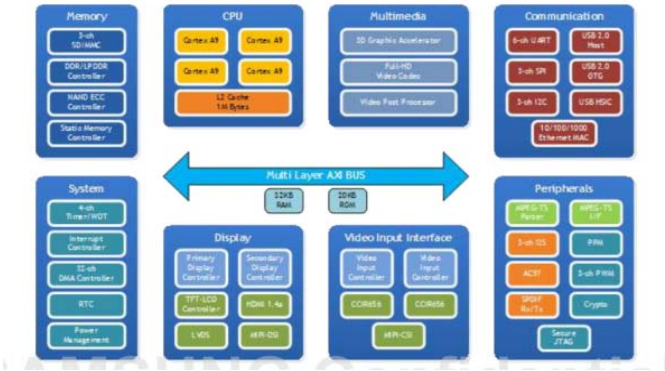


Figure 2. The block diagram of the SoCS5P4418

B. Laser Scanner

For measuring the distance between robots and unknown obstacles, the robot is equipped with laser scanner leishen LS01C, as shown in figure3.The laser scanner LS01C can achieve 360-degree and two-dimensional planar scanning within the range of 6m, producing point cloud map information used for mapping process.

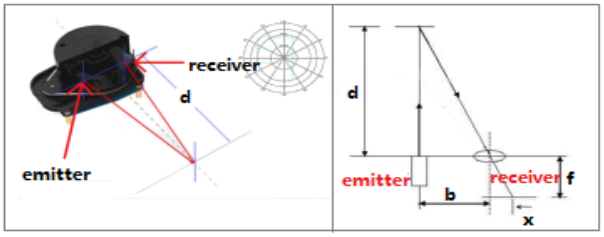
a) Laser Scanner, leishen LS01C b) The robot equiped with laser scanner

Figure 3. Laser Scanner

In the process of laser scanner, Laser Triangulation Method is used to measure the distance between the robot and the obstacles in the environment. As shown in figure4, certain amount of the laser will be emitted out of the scanner and then corresponding laser will be reflected by the surface of the obstacles if the laser hit the obstacles. Then the distance d between the laser and obstacles can be measured as follow:

$$d = (b \bullet f)/x \tag{1}$$

In this equation, b represents the distance between the axis of the emitted laser and the axis of the receiver and f represents the focal length of lens in the receiver. Then x measures the distance between the axis of the lens in the receiver and the point which is determined by the laser reflected and the focal length of lens.



a) The method applied in laser scanner leishen LS01C b) The Laser Triangulation Method

Figure 4. The Laser Triangulation Method

C. RobotOdomModel

In order to represent the position of robots in 2D environment, we adopt a 3D vector containing the robot's location (x, y) and orientation θ in our experiments. The location is measured by the robot x and y coordinates and the orientation is measured by the angle from the original orientation to current orientation. Then odometry is used in the experiments to estimate the robot position relative to a starting location. The position of robot can be estimated from the odometry as below:

$$pose = \begin{vmatrix} x \\ y \\ \theta \end{vmatrix} = \begin{vmatrix} \Delta s \bullet \cos\left(\theta + \frac{\Delta\theta}{2}\right) \\ \Delta s \bullet \sin\left(\theta + \frac{\Delta\theta}{2}\right) \\ \Delta\theta \end{vmatrix} \tag{2}$$

In this equation, Δs is measured by the distance from the last location to current location of the robot and $\Delta\theta$ represents the degree the robot has turn from the last orientation to current orientation.

III. GMAPPING ALGORITHM

The abstract concept of SLAM is the problem of estimating the potential robot trajectories and building a map given the robot's observations about the environment and its odometry measurements. Then the main idea of gmapping algorithm for SLAM is to evaluate a posterior $p(x_{1:t}, m \mid z_{1:t}, u_{0:t})$ over maps and robot's trajectories given robot's observations $z_{1:t}$ and odometry measurements $u_{0:t}$.

The gmapping algorithm is based on Rao-Blackwellized particle filer(RBPF)[3][4] and the key idea of RBPF for SLAM is to estimate a posterior $p(x_{1:t} \mid z_{1:t}, u_{0:t})$ about potential trajectories of the robot given its observations $z_{1:t}$ and

measurements $u_{0:t}$. Finally we can describe how gmapping algorithm overcomes the problem of SLAM as below:

$$p(x_{1:t},m|z_{1:t},u_{0:t})=p(m|x_{1:t},z_{1:t})p(x_{1:t}|z_{1:t},u_{0:t}) \quad (3)$$

In this equation, the robot pose is firstly estimated by computing posterior $p(x_{1:t}|z_{1:t},u_{0:t})$, then the map is built by computing the posterior $p(m|x_{1:t},z_{1:t})$ if the trajectories $x_{1:t}$ and the observations of robots $z_{1:t}$ are given. Following this strategy, each "particle" in gmapping algorithm carries a map where the pose of robot is known.

Then, based on the equation3, the gmapping algorithm can be described as the following steps:

A. Initialization

Initially, certain number of particles which are carried out with pose and maps are initiated around the robot. A 3D vector (x,y,θ) containing the robot's location and orientation is adopted to represent the pose and another 3D vector $(visit,count,(acc\_x,acc\_y))$ is carried with each grid of the 2D map, where $visit$ means that how many times the grids have been visited by laser, $count$ means that how many times the grids have been reached by the end of the laser and (acc_x,acc_y) means that the sum of the ends of lasers.

B. Processing the laser scan to build maps

In this step, the odometry information is given to transform the laser scan into the robot coordinate in which the origin is the initial pose of the robot. With the transformed laser scan, we can compute the 3D vector $(visit,count,(acc\_x,acc\_y))$ as following steps:

1) computing visited grid and marked grid: the grids in the 2D occupancy map which is scanned and the grids which the laser reaches are measured as shown in figure5. Bresenham algorithm[5] is used in this step to determine which grid is scanned by the laser. Following this strategy, $visit$ of the yellow grids and black grids will add one if a laser pass them and $count$ of the black grid will add one if the end of a laser reach them. Also the (acc_x,acc_y) of the black grid will add $(end\ \ x,end\ \ y)$.

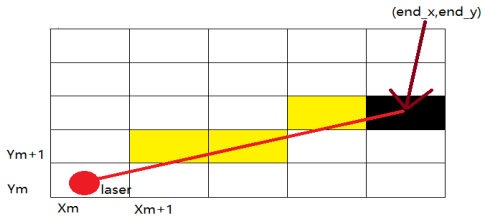


Figure 5. A 2D occupancy grid map built on one laser

2) computing the probability being obstacle of the grid: following last step, the visited times and the marked times of grids in the 2D occupancy map have been computed. Then in order to avoid being mistaken as obstacles, the grids whose probability of being obstacles is higher than a threshold is considered to be obstacles grid. Otherwise, the obstacles is considered to be the free space. Following this idea, the probability of the grids being obstacles can be determined as following equation:

$$p = n/visits \quad (4)$$

In this equation, n is equal to the marked times and $visit$ is equal to the visited times. Following this equation, the probability of the grid being obstacles increases if the marked times is higher and visited times is lower.

C. Updating the weight of the particles

In order to weigh how much the pose of the particles is close to the pose of the robot, gmapping algorithm uses the laser scan to match the maps carried with the particles. The score of the particles are updated as following steps:

1) computing the distance between obstacles and the ends of the laser: The grids whose probability of being obstacles is higher than the threshold is considered to be obstacles grid. Then the distance between the laser and the obstacle can be considered as the distance between points (end_x,end_y) and points $(acc\_x/visit,acc\_y/visit)$ of the nearest obstacle grid by the end of the laser:

$$d=(end\_x-acc\_x)^2+(end\_y-acc\_y)^2 \quad (5)$$

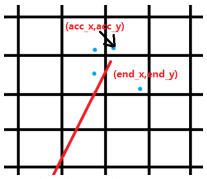


Figure 6. The method used to compute the distance

*2) computing the score: i*n the gmapping algorithm, the measure error of the observation from scan can be considered as a Gauss distribution with a mean value of 0 and a standard deviation of simga. Following this strategy, the score of each particle can be computed as follow:

$$score(d)=\exp\left(-\frac{1}{sigma\bullet d^2}\right) \quad (6)$$

In this equation, the lower the value of d is, the higher score the particle will get.

D. Updating the current pose of the robot

In the last step, the score of the particles is computed. Then in this step, the particle with the highest score is chosen to represent the pose of the robot.

E. Resampling

In the step of resampling, the particles with low importance weight are replaced by new sample with high weight. Then in order to avoid losing the diversity of the particles, some particles is added with some noise randomly.

IV. GMAPPING IMPLEMENTATION ON EMBEDDED SYSTEM

In this section, we will introduce the implementation of gmapping on the embedded system not using ROS so that the CPU consumption can reduce. Then in next section, we will improve its performance on the time consumption by improving the the architecture of the system.

A. Architecture of mapping system

As shown in figure7, the gmapping system is completed in three parts including LaserDriver, RobotDriver and GMappingProcess.The part LaserDriver is functioned to process the primitive laser scan and the part RobotDriver is functioned as the driver of the robot and to return the odometric information from robot to the GMappingProcess. As discussed in section III, the part GMappingProcess is functioned as to build maps and estimate pose of the robot.

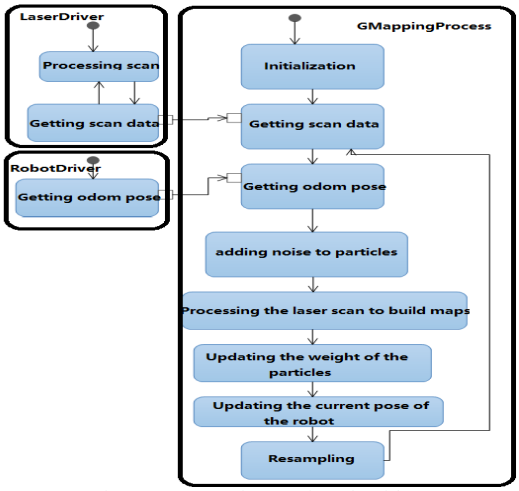


Figure 7. Steps of gmapping algorithm

B. Implementation on the NanoPi2

The gmapping system as discussed before is implemented on the NanoPi2. We implement the Thread of the LaserDrive Part in the system on core0, the RobotDriver Part on core1 and the GMappingProcess Part on core2. And these three thread communicate with each other through the shared memory. Finally, core3 is available for other development.

As shown in figure8, the flow works as follow: firstly, the laser scan comes in and the LaserDriver thread process the scan data into an array, which can be inputted into GMappingProcess. Then the odometric information of the robot comes in and the RobotDrive process it into pose of the robots. Receiving the data from laser scan and robot odometry, the thread GMappingProcess can build maps and estimate accurate pose of the robot.

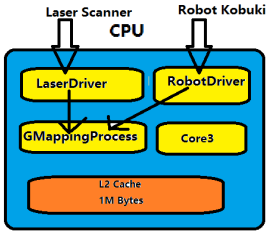


Figure 8. Implementation on NanoPi2 CPU

C. CPU consumption.

Using this system, CPU load needed for running ROS is not required so that the usage of CPU is more efficient. Additionally, the parallel execution for LaserDriver, RobotDrice and GMappingProcess reduce the time needed for running the system.

V. IMPROVING IMPLEMENTATION ON CPU

Now, we can look at what we can do to improve the performance. As discussed in the last section, we can note that the core3 in the CPU is available for us to develop. Then we can note that the thread GMappingProcess is suitable for parallel execution. So in this section, we will improve its performance by improving the architecture of the mapping system.

A. Improving architect of mapping system

The improving Mapping system is completed in four parts including LaserDrive, RobotDrive, MappingProcess and EstimatingPose. In the gmapping algorithm, the time consuming is from two steps including processing the laser to build maps and updating the weight of the particles. And actually these two steps are suitable for parallel execution for updating maps and pose between particles are independent. After finishing two steps, the particles with highest score can be chosen to represent the map and pose of the robot. Therefore, the mapping system can be improved as shown in figure9.

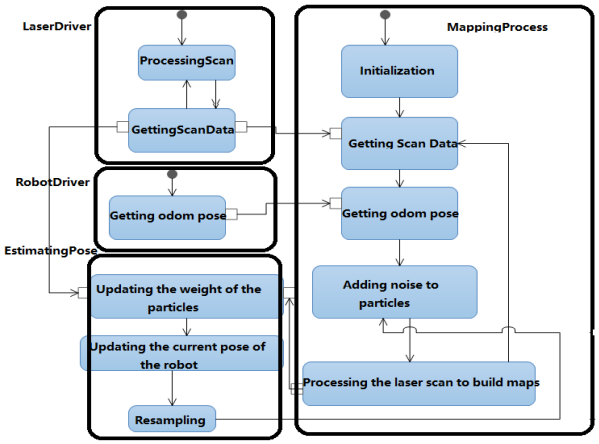


Figure 9. Steps of the improving mapping process

B. Improving implementation on the NanoPi2

We implement our improving mapping system on the CPU of NanoPi2 as follow. The improvement we make is to parallelize two steps of the GMappingProcess including processing laser to build maps and updating the weight of the particles for the core3 on the CPU on NanoPi2 is available.

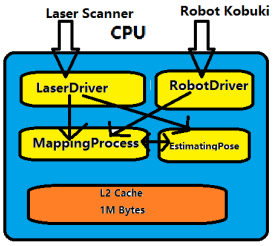


Figure 10. Improving implementation on NanoPi2 CPU

As shown in figure10, the improving flows works as follow: after the first particle finishes updating the map carried with it, the thread for updating weight of the particle will be triggered. At the same time, the second particle will update its map from the laser and odometric information and then update the weight. After the last particle finishes updating its weight, the particle with highest score will be chosen to represent the map and the pose of the robot.

C. Time consumption

Though the core3 is occupied by a new thread, the time consumption actually is reduced by improving the architecture of the mapping system. Because the part EstimatingPose and MappingProcess are suitable to be executed parallelly and the time cost for gmapping algorithm is mainly consist of the time of processing the laser to build and updating the weight of the particles.

We can consider the time used to process the laser for each particle as t_1 and the time used to update the weight of the particles as t_2. Then we can compute the time cost for not improving mapping system as follow:

$$t = n \bullet (t_1 + t_2) \tag{7}$$

In this equation, t is the estimated time for running gmapping algorithm.

In the improving mapping system, we can compute the time cost as follow:

$$t = n \bullet t_1 + t_2 \tag{8}$$

Compared to the last equation, we can note that the time required for running the gmapping algorithm is reduced.

As a result, the parallel execution for MappingProcess and EstimatingPose allows us to reduce the time required for building maps and estimating pose.

VI. Experiment

This section describes a practical implementation of the simultaneous localization and mapping on a embedded system not using ROS and the environment where the robot is operated. Then we present our experiment results and analysis on our system performance.

A. Experimental Setup

In this experiment, the robot is operated to explore the laboratory in our laboratory environment, as shown in figure10. Then a 2D occupancy map is built on the hardware development platform and in order to test the accuracy of the map, we visualize our data to test the accuracy of the map.

Figure 11. The laboratory where the experiments are done

Specifically, a hardware development platform NanoPi2 (Cortex-A9, 1G CPU,) running Ubuntu is utilized for robot teleoperation and mapping. Then, Kobuki robot and leishenLS01C laser range finder are controlled by NanoPi2 through USB interface. As NanoPi2 can control the robot and sensor, it can execute programs to build map from the laser scan and robot odometry information. The thread LaserDriver is run to process the scan. The thread RobotDriver is run to return the odometry information of the robot to the thread MapppingProcess and thread EstimatingPose.

In order to generate accurate map, we adjust the value of parameters that affect the performance of the GMappig algorithm. As shown in table I, the resolution of the map is 5cm and the range of the map is 40m×40m. The number of particles, one of the important parameters, is 30. With these settled parameters, we operate the robots and run the mapping process to build maps in the practical environment.

TABLE I. THE PARAMETERS USED IN THE GMAPPING ALGORITHM

Parameters	**value**
delta_	0.05
xmin_	-20.0
ymin_	-20.0
xmax_	20.0
ymax_	20.0
particles_	30

B. Mapping result:

1) the accuracy of the map: in the experiments, we run our robot in laboratory as shown in figure 11 and figure 12 to build the maps. As shown in the image of the figure13 and figure14, the final map is high-quality and the map actually match the environment of the laboratory.

Figure 12. The small court where the robot was run and figure 13 was created.

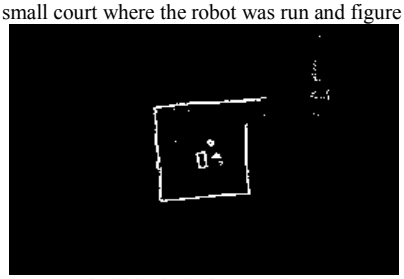


Figure 13. The 2D map built in the indoor environment in the laboratory as shown in figure 12 and drawn with openCV

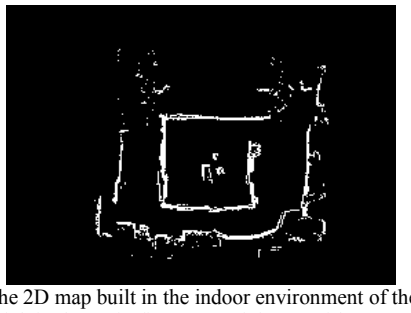


Figure 14. The 2D map built in the indoor environment of the laboratory which is shown in figure11 and drawn with openCV

2) The CPU load for mapping : as shown in table II, we compare the CPU load of mapping of different implementation. From the result, we can see that the CPU load required for executing our mapping process is relatively small when the time need for running robot and processing the laser are taken into account.

TABLE II. THE CPU LOAD OF MAPPING SYSTEM

Mapping system	Particles	CPU Load
ROS package	30	83%
Not using ROS	30	36%

3) The time required for mapping process: in the experiments, we measure the time needed for adding a serial of laser scan in different mapping system. As shown in tableIII, we can conclude that the improving mapping system take less time to add a scan to the map than the not improving mapping system.

TABLE III. THE TIME CONSUMPTION OF MAPPING SYSTEM

Mapping system	Time(ms)
Not improving mapping system	38
Improving mapping system	23

VII. CONCLUSION

In this paper, we present a successful implementation of mapping system on ARM board NanoPi2 not using ROS. In order to reduce the memory consumption and time consumption, we implement our mapping system on four parts by improving the architecture of the mapping system. One part is the LaserSensor program which processes the laser and returns the processed data to another two parts. Then with the odometry information from another part RobotOdometryModel, we run the GMappingProcess program including MappingProscess and EstimatingPose on the embedded system to build maps.

Then from the results carried out, one can conclude that our mapping system can achieve a good performance. It includes that the generated maps in given environment are high-quality and that the time required and the CPU loaded is relatively low. This allows us to do future research on our high performance ARM Board.

The first thing for future research is to increase the occupancy of the maps. Focusing on the parameters that affect the performance of the occupancy of the generated maps, we can optimize the parameters to generate more accurate maps with the presented mapping system. Another important future work is to implement other algorithms on the mapping system. Other algorithms like GRIDSLAM[7] and Unscented FastSLAM[8][9] that solve the SLAM problem can be implemented in the system we present in this paper. And future research can include automatic navigation. With the built 2D occupancy maps, navigation algorithms like dynamic window algorithm (DWA)[6] and D* Lite[10] can be considered to be implemented so that robot can achieve goals of automatic navigation.

ACKNOWLEDGMENT

This research work is supported by National Undergraduate Innovative and Entrepreneurial Training Program (No.201610561123). Guangdong province science and technology plan projects (2017A010101031). the Fundamental Research Funds for the Central Universities (2017MS048).

REFERENCES

[1] Grisetti, G., C. Stachniss, and W. Burgard. "Improved Techniques for Grid Mapping With Rao-Blackwellized Particle Filters." *IEEE Transactions on Robotics* 23.1(2007):34-46.

[2] Grisettiyz, G., C. Stachniss, and W. Burgard. "Improving Grid-based SLAM with Rao-Blackwellized Particle Filters by Adaptive Proposals and Selective Resampling." *IEEE International Conference on Robotics and Automation* IEEE, 2005:2432-2437.

[3] A. Doucet, J.F.G. de Freitas, K. Murphy, and S. Russel. Rao-Blackwellized partcile filtering for dynamic bayesian networks. In *Proc. Of the Conf. on Uncertainty in Artificial Intelligence (UAI)*, pages 176 – 183, Stanford, CA, USA, 2000.

[4] K. Murphy. Bayesian map learning in dynamic environments. In *Proc. of the Conf. on Neural Information Processing Systems (NIPS)*, pages 1015 – 1021, Denver, CO, USA, 1999.

[5] Bresenham, J. E. "Algorithm for Computer Control of a Digital Plotter." *Ibm Systems Journal* 4.1(1965):25-30.

[6] Fox, Dieter, W. Burgard, and S. Thrun. "The dynamic window approach to collision avoidance." *IEEE Robotics & Automation Magazine* 4.1(2002):23-33.

[7] Hähnel, D., et al. "A Highly Efficient FastSLAM Algorithm for Generating Cyclic Maps of Large-Scale Environments from Raw Laser Range Measurements." *Proc. of the Conference on Intelligent Robots and Systems* 2003.

[8] Kim, Chanki, R. Sakthivel, and K. C. Wan. "Unscented FastSLAM: A Robust and Efficient Solution to the SLAM Problem." *IEEE Transactions on Robotics* 24.4(2008):808-820.

[9] Kim, Chanki, H. Kim, and K. C. Wan. "Exactly Rao-Blackwellized unscented particle filters for SLAM." 19.6(2011):3589-3594.

[10] Koenig, S, and M. Likhachev. "D* Lite." *Aaai/iaai* (2002):476-483.

[11] Folkesson, J, and H. Christensen. "Graphical SLAM - a self-correcting map." *IEEE Intl. Conf. on Robotics and Automation* 2004:383--390.

[12] Eliazar, Austin, and R. Parr. "DP-SLAM: Fast, Robust Simultaneous Localization and Mapping Without."in *in Proc. 18th Int. Joint Conf. on Artificial Intelligence* (IJCAI-03, pp. 1135 – 1142, 2003.

[13] Vincke, B, A. Elouardi, and A. Lambert. "Design and evaluation of an embedded system based SLAM applications." *System Integration (SII), 2010 IEEE/SICE International Symposium on IEEE,* 2010:224 - 229.

[14] Vincke, Bastien, et al. "Efficient implementation of EKF-SLAM on a multi-core embedded system." 2.1(2012):3049-3054.