Maze Runner

Eric Marcoux
University of Massachusetts Lowell
Human Robotics Interaction Lab
eric_marcoux@student.uml.edu

Brian Day
University of Massachusetts Lowell
brian_day@student.uml.edu

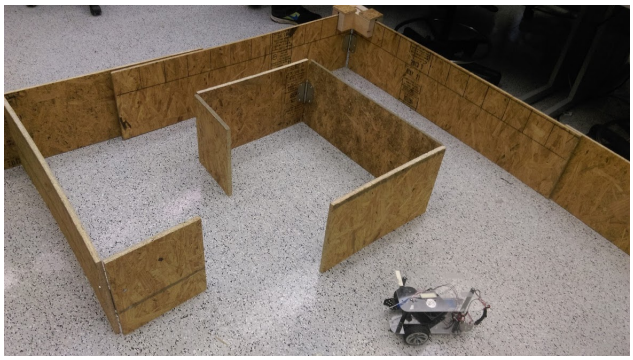

Figure 1. Robot Maze

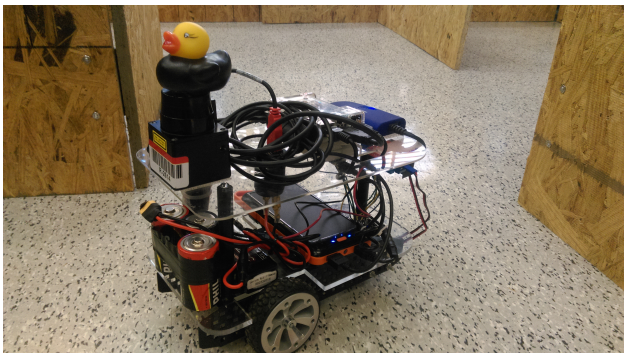

Figure 2. Robot Design

Author Keywords
maze navigation, SLAM, FastSLAM, odometry, robot, low cost

ABSTRACT
FastSLAM [1] is a technique utilized in robotics to build a map of an unknown, often indoor, environment probabilistically taking into account sensor and odometry error, while simultaneously localizing a robot in that map. In this paper we will describe the design and development of a low cost mobile robot that is able to perform a custom FastSLAM algorithm through the utilization of the Robot Operating System (ROS) [2]. First the design and development of the robot, the robots odometry system, and sensor array will be described. Then the design of the FastSLAM implementation will be visited.

INTRODUCTION
As stated previously, SLAM, or Simultaneous Localization And Mapping is a technique utilized in robotics to build a map of an environment, while simultaneously localizing a robot in that map. In this way it solves the main problem of robot mapping: to make a map you need to know the location of the robot, but to know the location of the robot you need to have a map to localize it in. As stated this paper will describe the design and development of a robot capable of such a feat and the development of one possible slam algorithm, known as FastSLAM [1]. To evaluate the robot system a variety of mazes were developed such as the one seen in Figure 1. The goal of the robot was to drive through the maze, build both a map of the maze, and localize itself within that map.

PROJECT DESCRIPTION

Robot Design

Mechanics and Electrical System

The construction of the robot can be seen in Figure 2 and is as follows. The main body is made up of two levels of acrylic sheets. Both of these sheets are 10”x6” with rounded corners. The bottom sheet is quarter inch and the top sheet is 1/16 inch. The smaller top sheet was chosen to help reduce the weight of the robot while the bottom was chosen for rigidity. Below the bottom sheet is a vex c-channel with pre-drilled holes. The front drive train is a differential drive powered by two VEX high strength plastic gear motors. The rear wheel is an omnidirectional wheel. This provides a castor like mechanism to allow the rear wheel to move freely with limited friction while avoiding castor lock which could potentially disrupt the odometery system. The front of the robot suffered from having limited weight over the front two wheels, affecting it’s ability to rotate in place. To compensate for this three D-Cell batteries were added to provide a weight-box.

To power the front two wheels an L298N H-bridge was used. An H-bridge is an electrical device used to interface low current control logic to high current devices, while being

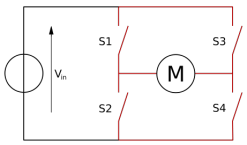

Figure 3. H-Bridge Circuit

able to output power in both directions. Its is traditionally implemented in terms of four high power switching mosfets, see Figure 3. The H-bridge[3] that was used utilized a 6 pin control system. There was an Enable pin for motors A and B, and four Input pins that could be used to control the polarity of each of the individual motors. To drive the motor in a certain direction one of the two Input pins for the desired motor was set HIGH (1), and the other LOW (0). The enable pin was then controlled with pulse-width-modulation (PWM) to set the speed at which the motor drove. Power to the H-Bridge was provided by a 2200mAh 3S 25C model aircraft LIPO battery, where 3S means three 3.3v cells (11.4V) and 25C means that the peak power output of the battery is twenty-five times the current rating.

To control the H-bridge an Arduino Nano[4] was programmed using an interface called ROSSerial to communicate with a Raspberry PI over a USB serial connection. The Raspberry PI was running a software stack known as the Robot Operating System, or ROS [2]. ROS is a robot platform designed around the concept of multi-process message passing. Under this model it is possible to separate the drivers for a given robot from the algorithms that control that robot. In this way it is possible to reuse existing code without necessarily understanding how it works under the hood. By running ROS on the Raspberry PI we were able to take advantage of a large amount of tooling, and the necessary algorithms described hererin.

To power the RaspberryPI a second, 15,000mAh cell phone charger, battery was used. This battery provided a 2.1A, 5V, usb connection to the RaspberryPI and a 1A, 5V, power connection to the distance sensors described later. Proir to this, in an earlier iteration, the RaspberryPI and other sensors received power over a car cell phone charger attached to the model aircraft LIPO battery. This charger was unable to provide the necessary current as it was falsely advertised. As the H-Bridge described earlier requires a common ground between the control logic and the power bus the arduino Nano's ground reference pin was shorted to the ground of the LIPO battery at the screw terminal on the H-bridge.

Sensor Suite and Odometry

One of the essential parts of a robot attempting to perform FastSLAM is a good sensor suite. This sensor suite is required for two of the major components of SLAM - the motion model (odometry) and the sensor model. The sensor model refers to a set of distance measurements (often implemented with a laser scan) to actually sense the robot's environment and build to map. The motion model is used to predict the position of the robot in the map.

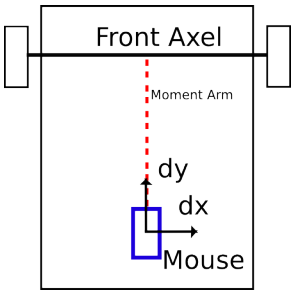

Figure 4. Single Mouse Odometry

In the first revision of the robot this sensor suite included a hokuyo urg laser[5] for range information, and a single mouse to provide odometry. This was inspired by the article "High-Precision Robot Odometry Using an Array of Optical Mice" [6]. The logic behind the choice of a mouse is that, in Linux, by subscribing to $/dev/input/mouseN$ one is able to quickly and easily get a position delta in both the X and Y direction of the mouse. By mounting the mouse at some known moment arm from the center of the axis of rotation it is possible to determine the motion travel of the robot, as seen in Figure 4. By differentiating dy with respect to time it is possible to get the linear velocity. To get the angular velocity one can simply solve to find that $v_t = \omega r \implies \omega = (dy/dt)r$ where r is the length of the moment arm. The problem we ran into with this approach is that the inconsistencies in the surface of the floor cause poor sensor performance.

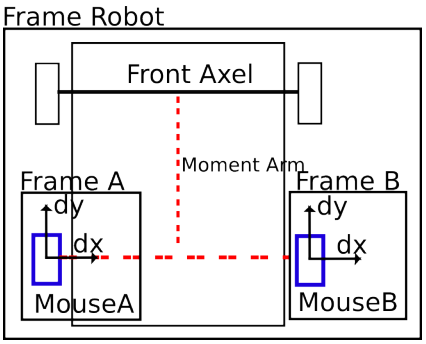

Figure 5. Dual Mouse Odometry

The second revision of the sensor suite was a direct implementation of "High-Precision Robot Odometry Using an Array of Optical Mice" [6] which takes into account this shortcoming of mice by using a system of linear equations using multiple mice. The diagram for which can be seen in Figure 5. The system of linear equations can be represented by the below equation, where ΔX_R is the change in the real x position of the robot, ΔY_R is the change in the real y position of the robot, and $\Delta\omega$ is the change in rotation of the robot. Also α_i is the angle between the center of the front axle and the i^{th} mouse frame, ϕ_i is the angle to take that

rotated frame and make it's y axis line up with the y axis of the robot frame, and r_i is the distance between the center of the front axle and the i^{th} mouse frame.

$$\begin{bmatrix}\Delta X_R \\ \Delta Y_R \\ \Delta\omega\end{bmatrix} = A^{+}\begin{bmatrix}dx_1 \\ dy_1 \\ dx_2 \\ dy_2\end{bmatrix}$$

where

$$A = \begin{bmatrix}sin\alpha_1 & -cos\alpha_1 & r_1 cos\phi_1 \\ cos\alpha_1 & sin\alpha_1 & r_1 sin\phi_1 \\ sin\alpha_2 & -cos\alpha_2 & r_2 cos\phi_2 \\ cos\alpha_2 & sin\alpha_2 & r_2 sin\phi_2\end{bmatrix}$$

and A+ is the Moore-Penrose pseudoinverse of A

$$A^{+} = A^{T}(AA^{T})^{-1}$$

Unfortunately this method did not work either for similar reasons to iteration one. To truely take advantage of the work done in "High-Precision Robot Odometry Using an Array of Optical Mice" [6] it would be necessary to increase the number of mice to a greater number. For example in their work six mice were used to achieve accurate results. Due to the size of the robot and the limited number of available mice this was not an option.

The third iteration of the robot relied on another odometry technique entirely, visual odometry. Visual odometry is a technique that utilizes optical flow between feature points, often acquired using SURF, SIFT or FAST, to determine relative motion within a two dimensional image. This motion is then correlated with a depth image to convert from image space into world space, thereby giving real world odometry[7]. To do this we utilized a PrimeSense Carbine, a sensor similar to, through more sophisticated than, a Microsoft Kinect, to act as an RGB-D camera. In using this device it was possible to remove the requirement of the hokuyo lidar by simply taking a single scan line of the disparity frame[8] of the camera as a pseudo-laser scan. Then both this disparity frame and the camera's RGB frame were handed over to the visual odometry system to estimate motion. Two visual odometry packages, available in ROS, were tried. The first was "fovis_ros" which takes into account the depth frame as described above. The second was "viso2_ros" which instead estimated position without a depth frame by using height of the camera off the ground plane and the position of that plane in the image to scale the image's relative motion to real world motion. Unfortunately both of these vision algorithms did not work well within the maze due to the limited number of feature points available to the vision algorithm and the high computational complexity of doing these calculation. The high computational complexity then forced the off-boarding of odometry to a secondary computer. This secondary computer was limited in it's network throughput because of the RaspberryPI's 54Mb wireless connection. This caused too high of a frame loss to effectively send both depth and vision meaning that "fovis_ros" would not work. Theora and JPEG compression between the PI and remote host was attempted to alleviate the network traffic but this increased load on the RaspberryPI leading to the same problem. The package "vision2_ros" which relied only on the RGB frame was able to run but suffered from odometry loss during periods of rapid rotation, especially with the limited key points described earlier.

The final iteration of the robot's sensor suite returned to only having a laser range finder. To get the odometry the input command velocity was simply integrated and returned as the robot odometry. To make sure this odometry was as accurate as possible it was necessary to calculate the torque curve of the two vex high strength motors. This was done by measuring the distance travelled by the robot over a certain period of time at varying input powers. These values were then plotted. By doing a least squares fitting [9] as seen below it's possible to determine the exponential distribution.

$$\begin{aligned} y &= Ae^{Bx} \\ lny &= lnA + Bx \end{aligned} \tag{1}$$

By applying least-squares

$$\begin{aligned} a &= \frac{\sum_{i=1}^{n} lny_i \sum_{i=1}^{n} x_i^2 - \sum_{i=1}^{n} lnx_i \sum_{i=1}^{n} x_i lny_i}{n\sum_{i=1}^{n} x_i^2 - (\sum_{i=1}^{n} x_i)^2} \\ b &= \frac{n\sum_{i=1}^{n} x_i lny_i - \sum_{i=1}^{n} x_i \sum_{i=1}^{n} lny_i}{n\sum_{i=1}^{n} x_i^2 - (\sum_{i=1}^{n} x_i)^2} \end{aligned} \tag{2}$$

where $A = e^a$ and $B = b$.

From this we determined the motors follow the exponential distribution of $power = 0.090149e^{6.44209*speed}$. To get the normal distribution of possible values we use a euclidean distance like formula as was done in the homework of $\sigma = \sqrt{v_{linear}^2 + \omega^2}$ centered on the commanded velocity. To compensate for this less than ideal odometry scan matching[10] was extensively used as described in the FastSLAM Algorithm section.

FastSLAM Algorithm

Implementation of FastSlam:

$$\begin{aligned}
&\textbf{Algorithm FastSlam_occupancy_grids}(\{X_{t-1}\}, u_t, z_t): \\
&\quad \bar{X}_t = X_t = 0 \\
&\quad for\ k\ =\ 1\ to\ M\ do \\
&\quad\quad x_t^{[k]} = \textbf{sample_motion_model}() \\
&\quad\quad w_t^{[k]} = \textbf{measurement_model_map}() \\
&\quad\quad w_t^{[m]} = \textbf{updated_occupancy_grid}() \\
&\quad\quad \bar{X}_t = \bar{X}_t + \langle\rangle x_t^{[k]}, w_t^{[k]}, w_t^{[m]} \\
&\quad endfor \\
&\quad for\ k = 1\ to\ Mdo \\
&\quad\quad draw\ i\ with\ probability\ \propto w_t^{[k]} \\
&\quad\quad add \langle\rangle x_t^{[k]}, m_t^{[k]}\ to X_t \\
&\quad endfor \\
&\quad return X_t
\end{aligned} \tag{3}$$

[11, p.479]

Algorithm occupancy_grid_mapping$(\{l_{t-1,i}\}, x_t, z_t)$

$if\ \mathbf{m}_i\ in\ perceptual\ field\ of\ z_t\ then$
$l_{t,i} = l_{t-1,i} + \mathbf{inverse\_sensor\_model}(\mathbf{m}_i, x_t, z_t) - l_0$
$else$
$l_{t,i} = l_{t-1,i}$
$endif$
$endfor$
$return\{l_{t,i}\}$

(4)

[11, p.286]

Algorithm inverse_sensor_model()

$Let x_i, y_i be the center - of - mass of \mathbf{m}_i$
$r = \sqrt{(x_i - x)^2 + (y_i - y)^2}$
$\phi = atan2(y_i - y, x_i - x) - \theta$
$k = argmin_j |\phi - \theta_{j,sens}|$
$if\ r > min(z_{max}, z_t^k + \alpha/2)\ or |\phi - \theta_{k,sens}| > \beta/2\ then$
$return\ l_0$
$if z_t^k < z_{max}\ and\ |r - z_t^k| < \alpha/2$
$return\ l_{occ}$
$if\ r <= z_t^k$
$return\ l_{free}$
$endif$

(5)

[11, p.288]

ANALYSIS OF RESULTS

In an effort to analyze the performance of our "maze runner" it was necessary to develop an obstacle course, as can be seen in Figure 1. This maze was made out of plywood, making it possible to dynamically change the maze between runs. This was done to ensure that the mapping algorithm did not suffer from probabilistic over fitting to a single "world".

To develop a baseline to compare our FastSLAM implementation against, each maze was first run with the mapping algorithm gMapping [12], which, as described on their website, is "...a highly efficient Rao-Blackwellized particle filter to learn grid maps..." This package is the go-to standard for SLAM localization on the ROS platform. Another well-known SLAM package is hector-slam [13] which is similar to gMapping with the exception that it does not rely on direct odometery information but rather only on high-frequency scan-matching. The logic behind this is that sensors such as the hokuyo laser range finder that we had used are very particle dense and could be used as the sole source of odometry. However, in many cases gMapping is still chosen over this approach as hector slam is subject to localization loss during periods of rapid rotation. Due to the poor odometry model of the robot this effect will be seen regardless, as scan-matching will be more heavily relied upon than the odometry.

Figures 6, 7, and 8 show maps generated by both our FastSLAM implementation and gMapping. Unfortunately the maps generated during the first run were corrupted. On the second run gMapping failed on four separate attempts to build anything resembling a map. Our FastSLAM implementation however successfully built the map seen in Figure 6. This map is an exact likeness of the maze that it ran in and dimensionally, when compared with a measuring tape, matched within two hundred millimeters. On the third test we were able to get an accurate comparison between a well known and accepted mapping algorithm and our implementation of FastSLAM as seen in Figures 7 and 8. The first thing that is noticeable is the corruption of the lower half of the map in our FastSLAM implementation and the corruption

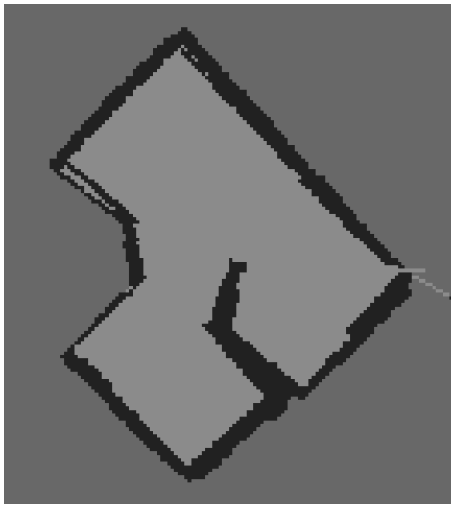

Figure 6. Second Run - Our FastSLAM

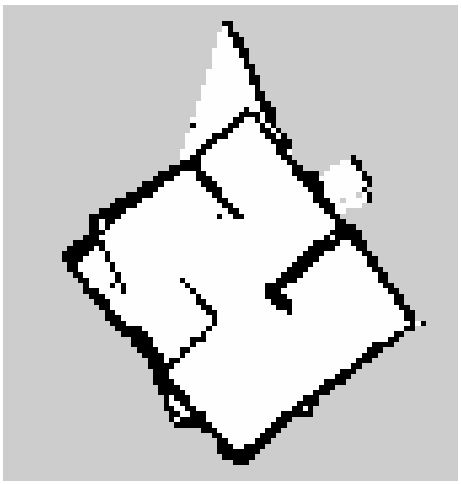

Figure 7. Third Run - Gmapping

in the top half of the gMapping test. These are both believed to be the result of inaccurate scan-matching as a result of the small size of the map and the rapid rotation that was necessary to get out of the top right room. The difference in where in the map it failed was due to a difference in the order of the visitation of the rooms. The second thing to notice is that gMapping's map's unoccupied areas, in other words the areas that are not unexplored and not walls, are far brighter than in our map. This is due to the fact that the gMapping algorithm had greater certainty of the map it was publishing than our FastSLAM implementation.

DISCUSSION

The work in this paper overlapped multiple disciplines in an attempt to achieve the goal of robot navigation and localization. The first, obvious, lesson was in the importance of good odometry. Odometry is something that's taken for granted in simulation as it's failure rate can be simply probabilistically modeled off of a given input velocity. Having a robot move at a commanded velocity in all situations, can realistically only happen in simulation. To compensate for this it is necessary to have some form of sensor suite in which one can probabilistically estimate the position and velocity of the robot post input control. To do this, this paper attempted the following methods: mouse based motion tracking, visual odometry, and direct dead-reckoning.

While great in theory mouse based motion tracking falls short due to an

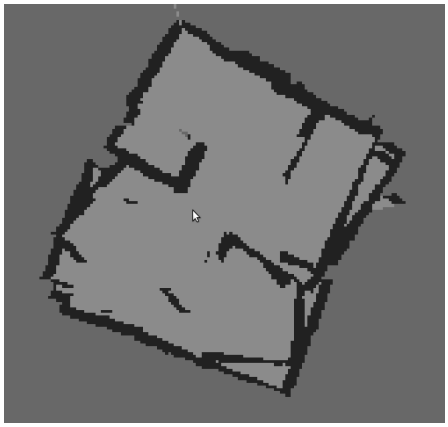

Figure 8. Third Run - Our FastSLAM

unexpected systemic error. The floor in which the maze was located is dotted with black markings. Due to the optics of a mouse, the mouse will cease to act as a mouse and begin to act as a line follower disrupting all distance readings. No amount of probabilistic modeling of the mouse's position will work in this case.

Visual odometry failed to work as it was highly susceptible to failure as a result of rapid rotation, which was common due to the small size of the mazes relative to the size of the robot we were working with. Should the image that the camera receives rotate at a rapid rate relative to the prior image the feature points between the two images will have minimal correlation meaning that the pose can not be estimated. Again no amount of probabilistic modeling can compensate for systemic error.

It was therefore decided to get odometry by simply dead-reckoning the position. From there it would be necessary to move the problem of odometry into the SLAM algorithm in the form of a really good scan matcher. Luckily we were able to rely on previous work and used an iterative closest point scan matcher that was available as a ROS package for this pose estimate. In doing so we were able to take advantage of the most expensive and powerful sensor on the robot to get our odometry, the Hokuyo laser range finder.

Integration of all the parts of the algorithm was the largest challenge on the software size. Parts of the algorithm needed to be heavily modified for it to work with the current iteration of the project. Some parts had to be swapped out due to bad performance with ROS packages to allow the project to continue. Updating the occupancy grid map was accomplished using algorithms (4) and (5), the struggle came from figuring out all the pieces and putting them together. Issues with the robot turning too quickly will cause bad mapping and by limiting the angular velocity on the robot we were able to help limit such issue.

CONCLUSIONS

In conclusion, through the robot system described within this paper, our team was able to perform SLAM in a real world environment. In this environment it was possible to rearrange the maze and quickly rebuild a new map of the world. From this we succeeded at our goal of building a low cost, with the exception of the LIDAR, robot able to do indoor localization and mapping. If this project was to be reimplemented investing in VEX encoders to generate better odometry would have been considered, rather than spending time researching and evaluating alternative odometry methods. Working with the poor odometry model we had, scan-matching became one of the most essential components of our robot's ability to map the environment. To fully complete a full FastSlam algorithm would a much larger time-frame and/or a larger team.

Presentation can be found at https://docs.google.com/presentation/d/1y6963UIUhwRkxvUNy972GxnduVCJ6lN$_8$2$QtaUknhbQ/edit?usp = sharing$

ACKNOWLEDGEMENTS

The work described in this paper was conducted as part of a Fall 2015 Mobile Robotics course, taught in the Computer Science department of the University of Massachusetts Lowell by Prof. Fred Martin.

The work load for the project described in this paper was distributed as follows. Eric Marcoux was in charge of the design and development of the robot, the design and development of the odometry system, and the sensor suite on the robot. Eric was also in charge of developing all of the launch file, other configuration, and research of existing ROS packages to fill in the pieces of the robot which were outside of the scope of the project. Eric also assisted in the motion model for the FastSLAM algorithm. Brian Day was in charge of developing the remainder of the FastSLAM algorithm, adaptation of Eric Marcoux's particle filter and adapting the uml_mcl package to work within the project for testing.

REFERENCES

1. M. Montemerlo, S. Thrun, D. Koller, B. Wegbreit, *et al.*, "Fastslam: A factored solution to the simultaneous localization and mapping problem," in *AAAI/IAAI*, pp. 593–598, 2002.
2. M. Quigley, K. Conley, B. P. Gerkey, J. Faust, T. Foote, J. Leibs, R. Wheeler, and A. Y. Ng, "Ros: an open-source robot operating system," in *ICRA Workshop on Open Source Software*, 2009.
3. "Tutorial - l298n dual motor controller module 2a and arduino." http://tronixlabs.com/news/tutorial-l298n-dual-motor-controller-module-2a-and-arduino/.
4. "Arduino nano." https://www.arduino.cc/en/Main/ArduinoBoardNano.
5. "Scanning range finder (sokuiki sensor)." https://www.hokuyo-aut.jp/02sensor/07scanner/urg_04lx_ug01.html.
6. S. Bell, "High-precision robot odometry using an array of optical mice," *2011 IEEE Region 5 Student Paper Contest*, 2011.
7. A. S. Huang, A. Bachrach, P. Henry, M. Krainin, D. Maturana, D. Fox, and N. Roy, "Visual odometry and mapping for autonomous flight using an rgb-d camera," in *International Symposium on Robotics Research (ISRR)*, pp. 1–16, 2011.
8. D. Tzovaras, N. Grammalidis, and M. G. Strintzis, "Disparity field and depth map coding for multiview 3d image generation," *Signal Processing: Image Communication*, vol. 11, no. 3, pp. 205–230, 1998.
9. E. W. Weisstein, "Least squares fitting–exponential." http://mathworld.wolfram.com/LeastSquaresFittingExponential.html.
10. A. Censi, "An icp variant using a point-to-line metric," in *Robotics and Automation, 2008. ICRA 2008. IEEE International Conference on*, pp. 19–25, IEEE, 2008.
11. W. B. Sebastian Thrun and D. Fox, "Probabilistic robotics," in *Probabilistic ROBOTICS*, pp. 286–288,486, MIT, 2006.
12. C. Stachniss and G. Grisetti, "Gmapping project at openslam. org," 2007.
13. S. Kohlbrecher, J. Meyer, O. von Stryk, and U. Klingauf, "A flexible and scalable slam system with full 3d motion estimation," in *Proc. IEEE International Symposium on Safety, Security and Rescue Robotics (SSRR)*, IEEE, November 2011.