Mapping with Depth Camera

Carolina Costa 84022, Francisco Melo 84053, Raul Vaqueiro 84175, Rodrigo Rego 89213

***Abstract*—This article is the report of the project that was developed by students of Autonomous Systems of the first semester of the year 2019/2020 of Instituto Superior Técnico. The objective of the project was to develop a program that, using an Occupancy Grid mapping algorithm, gives us a map of a static space, given the P3-DX Pioneer Robot's localization and the data from an Xbox Kinect depth camera. All the project was developed using ROS environment and packages. To achieve the major goal, it was used the Bresenham's Line Algorithm to implement our mapping algorithm as well as the Adaptive Monte Carlo Localization for the robot's localization. To evaluate our performance, it was used error metrics such as the binary difference of a reference map that was obtained using a ROS package (gmapping) and the one that our algorithm gives us. Another thing that should be mentioned is the fact that while acquiring the map, our reference frame is static, so when the algorithm is running, the map runs out this reference frame if we do not ensure a sufficiently high map dimensions. One solution for that is the use of a dynamic reference frame that is built along with the map.**

***Index Terms*—Mapping, ROS, Occupancy Grid Mapping Algorithm, Bresenham's Line Algorithm, Pionner robot, Kinect Depth Camera.**

I. Introduction

In robotics, it is crucial to have a map to execute a set of tasks in our daily routine. A map, in general, gives information about objects, regions, and specific places in the world.

The problem of robotic mapping is that of acquiring a spatial model of a robot's environment. The use of a map allows the robot to plan its movements, which is essential for the robot's actions such as localization, navigation (moving the robot to a specific point of the environment), or motion planning.

Acquiring maps with mobile robots is a challenging problem for several reasons. For example, the hypothesis space, that is, the space of all possible maps is enormous. Since maps are defined over a continuous space, the space of all maps has infinitely many dimensions.

Robots must possess sensors that enable it to perceive the outside world, in the case of this report, a depth camera, to acquire a map. Also, it is necessary to know the pose of the robot in every instance.

The problem of generating consistent maps from noisy and uncertain measurement data, under the assumption that the robot pose is known, is solved by the Occupancy Grid Mapping method.

Taking all of this into account, the objective of this work is to do mapping with a Kinect Depth Camera sensor and using the Pioneer robot.

II. Methods and Algorithms

Occupancy grid algorithms are a popular family of mapping algorithms. In this project, an occupancy grid mapping algorithm is used to map the desired 2D environments. To achieve this, the environment is divided in a grid with a finite number of square cells - all with the same size. The basic concept is to update the occupancy probability assigned to each cell that falls into the sensor cone of measurement z_t, for every iteration t in time and starting from a prior.

A. Occupancy Grid Mapping Algorithm

The gold standard of any occupancy grid mapping algorithm is to calculate the posterior over the map, given the data [1], as in

$$p\left(m|z_{1:t}, x_{1:t}\right). \tag{1}$$

The whole map is represented by m, $z_{1:t}$ is the set of all measurements up to the time iteration t, and $x_{1:t}$ is the sequence of all the robot's poses. This suggests that the path of robot is given by a localization algorithm.

The sensor cone of measurement $z_{1:t}$ is obtained after transforming the depth images given by the kinect sensor to a `LaserScan` structure - the middle row of the depth image is transformed into a `LaserScan` vector. The pose of the robot x, in a certain time instant t, is defined as $x = [x, y, \psi]^T$ (the 2D position x, y and the orientation angle of yaw ψ).

The posterior in equation (1) defines the estimation problem - when data is uncertain this is how the uncertainty is accounted for. As a result, the output should be the map, m, that maximizes the posterior, as in

$$m_e = \arg\max_m \quad p\left(m|z_{1:t}, x_{1:t}\right). \tag{2}$$

The standard approach is to break down the estimation problem given by equation (1), into one of estimating for all grid cells $\mathbf{m}_i$.

$$p\left(\mathbf{m}_i|z_{1:t}, x_{1:t}\right) \tag{3}$$

Each $\mathbf{m}_i$ has a binary occupancy value assigned to it - "1" for occupied and "0" for free.

Assuming that correlation between neighboring cells does not exist, because the computation of the posterior might be intractable otherwise, the posterior is therefore approximated as the product of its marginals:

$$p\left(m|z_{1:t}, x_{1:t}\right) = \prod_i p\left(\mathbf{m}_i|z_{1:t}, x_{1:t}\right). \tag{4}$$

To calculate $p\left(\mathbf{m}_i|z_{1:t}, x_{1:t}\right)$, each cell is updated using a Binary Bayes Filter [1], for every iteration t - thus, taking advantage of the binary state of each cell. This step uses the log-odds representation, so that numerical instabilities for probabilities near zero or one can be avoided.

The log-odd state $l_{i,t}$ of a certain cell i being occupied, with respect to the iteration t, is formulated as

$$l_{i,t} = l_{t-1,i} + \text{InverseSensorModel}\left(\mathbf{m}_i, z_t, x_t\right) - l_0. \tag{5}$$

The Inverse Sensor Model updates the occupancy value of the cells that fall into the sensor cone of measurement z_t. Otherwise, the occupancy value remains unchanged. The l_0 constant represents the prior of occupancy, given by the log-odds ratio:

$$l_0 = \log \frac{p(\mathbf{m}_i = 1)}{p(\mathbf{m}_i = 0)} = \log \frac{p(\mathbf{m}_i)}{1 - p(\mathbf{m}_i)}. \quad (6)$$

B. Inverse Sensor Model - Bresenham's Line Algorithm

The update of the occupancy values of a certain cell is done using the Bresenham's Line Algorithm [2].

The data obtained from a single row of a depth image, taken by a depth camera, resembles data that can be obtained from a laser rangefinder, where for each measurement beam k (or pixel) - belonging to the measurement cone z_t - a distance value is read. In these cases, the measurement beam k is very narrow with respect to the dimension of the individual cells: 2×2 cm^2, as opposed to what happens with sonar sensors.

As a result, it is important to have a way of determining which cells are traversed by a certain measurement beam k, belonging to the sensor cone of the depth camera, in order to determine which cells are considered for the occupancy value update - Inverse Sensor Model.

This problem resembles one of drawing lines, with various inclinations, in a grid that is composed of cells. For that use the Bresenham's Line Algorithm is a good choice, often used in computer graphics and implemented in firmware or in the graphics hardware of modern graphics cards due to its simplicity [2].

Therefore, given a measurement of a certain beam, the Bresenham's Line Algorithm can be implemented to select which cells are traversed by the measurement beam, and thus, which cells are considered in the Inverse Sensor Model, knowing the robot's pose. After that, all the cells traversed up to the measured depth should be considered as being free, and all the cells within the measurement uncertainty, represented by an initially defined object's thickness parameter (α), should be considered as being occupied.

C. ROS Packages

The following ROS packages were used to develop the project, and a brief description of each of them is presented below:

1) ROSARIA [3]: Provides a ROS interface for Piooner 3 robot [4]. It gives information about the robot pose, velocity, and acceleration control and sets the desired velocity and other commands.

2) urg_node [5]: Acquires data from the Laser Range Finder.

3) teleop_twist_keyboard [6]: Keyboard teleoperation for twist robots. It was used for robot navigation.

4) amcl [7]: Probabilistic localization system for a robot moving in 2D. It implements the adaptive Monte Carlo localization method, which utilizes a particle filter to track the pose of the robot in a known map. This package was used to localize the robot using the reference map produced by the gmapping package.

5) gmapping [8]: Implements a laser-based Simultaneous Localization and Mapping (SLAM). Creates a 2D occupancy grid map from laser and pose data collected by the robot. This package was used to make the reference map that is going to be used by the amcl package.

6) map_server [9]: Provides data as a ROS Service. Implements the ROS node `map_server` that offers a map via a ROS Service, and the ROS node `map_saver` that saves a map to a disk. The `map_saver` node was used to save the resulting map from the gmapping, and the `map_server` was used to offer this map as a ROS topic for amcl.

7) freenect_camera [10]: ROS driver for the Microsoft Kinect. Acquires data from the Kinect depth camera.

8) depthimage_to_laserscan [11]: Generates a 2D laser scan from a depth image.

9) rosbag [12]: Provides a command-line tool for working with bags. A bag is a file format in ROS for storing ROS message data. It is used in the project to store data acquisitions instead of running the algorithm in the real robot. This is useful for testing different implementations of the algorithm and thus increasing productivity.

III. Implementation

The implementation process of the solution developed for mapping with depth camera problem could be represented by the Fig. 1.

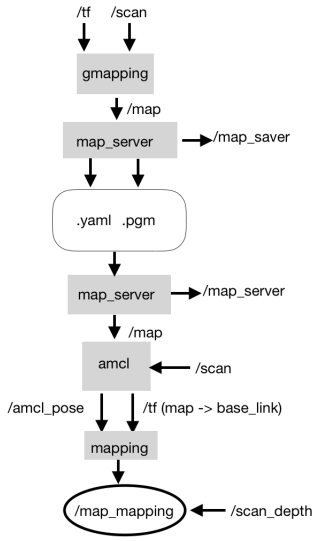

Fig. 1. Steps used to implement the project.

The processes start with the setup of the Pioneer robot's and Lidar laser in order to execute the first block of the Fig. 1, which represent the gmapping process. For future development, the process for data acquire is recorded to a rosbag using the record functionality of ROS.

Using the rosbag obtained, a map was created along with *gmapping* and *map_server*, resulting two files with <map>.yaml and <map>.pgm extension, where <map> is the name of the file, corresponding to the second block. The <map>.yaml have the metadata of the map and the

<map>.pgm file have the graphical representation of the map. The created map with this process is the map that is used as a reference to evaluate the performance of the algorithm develop in this project.

After the map is obtained as was referred previously, is necessary to have the information about the robot's localization. For that, the first step passes by read the map from the appropriate files (<map>.yaml and <map>.pgm) and using *map_server*, corresponding to the third grey block, the topic */map* is available for use by *amcl* package.

The fourth grey block corresponds to the *amcl* package, known as the one that can give the robot's localization, that it is assumed as an absolute localization and the data from the sensor is given by */scan* topic from the laser sensor. The robot's localization is given in two ways: */amcl_pose* and the */tf (map to base_link)*.

Finished the localization process, the setup of the Kinect depth camera is launch, in order to have all the necessary information to execute the developed node with the implementation of the Occupancy grid mapping algorithm already explained.

In the end, the mapping node, represented by the last grey block, return the */map_mapping* topic, which is the map obtained with the develop algorithm.

A. Setup of Pioneer

As was already referred, the robot used for implementation proposed is a Pioneer 3-DX by Mobile Robots. This robot is small lightweight two-wheel two-motor differential drive robot, considered as a monocycle robot. The main base of the robot are the two-motor wheels, the extra unmotorized wheel and the sonar system usually used to acquire data about the environment. Besides the information given by the sonar system, is also given information about the odometry of the robot, given at a 10 Hz rate. For communication and robot control proposes is used the *rosaria* and *teleop_twist_keyboard* packages. Instead of the sonar system, for this project is used a LIDAR Laser or a Kinect coupled to the robot as sensor for data acquisition about environment.

B. Setup of Lidar Laser

The LIDAR Laser sensor is used for the data acquire process for gmapping method, call on package *urg_node* given origin to the /scan topic. This sensor have an high accuracy, high resolution and wide angle according to the measure area of 240°from 20 to 5600mm and the step angle of approx. 0.36°(360°/1,024 steps), which are important characteristics for autonomous robots moving in unknown environment.

Plus this information is also necessary to compute the transformation between the laser and the base_link coordinate frame over time and for that was used the TF package.

C. Setup of Kinect Sensor

The *freenect* and *depth image to laser scan* packages were used for the setup of the Kinect depth camera generating the */scan_depth* topic one of the necessary information to execute the developed node with the implementation of the Occupancy grid mapping algorithm.

About the general Kinect, is composed by a infrared light projector, a depth sensor, a RGB camera, and a multi-array microphone. For this project proposes only the depth sensor is used for the environment data acquirement. The depth sensor range is from 0.8 m to 6.0 m with the vertical viewing angle 43°and horizontal viewing angle 57°and the acquirement frequency is 30 Hz which means that the maximal frame rate is 30 frames per second.

Notice that, a selection of the data acquired was implemented, when the depth image was adapted to the laser scan format namely, considering the depth image as a matrix, only the center line of the matrix was selected as crucial information. This is important because, for this project implementation the results would be different according the the selected line of the depth image. Also the maximum range used to the acquirement process where change to 5.0 m in order to minimize the noise over measures.

D. Data acquirement

While the acquirement process of the data for the rosbag, some rules are applied to the trajectory of the robot:

1) Zigzag Trajectory: In order to obtain a high precision map, a zigzag trajectory was used to overcome the sensor range limitation. In this way a higher range of zones were inspected by the sensor thus obtaining better performance, since unexamined zones on the map, meaning where the sensor had no range, are eliminated. Those zones are typically long corridor where the sensor made no acquisition to exceed the range.

2) Characteristic zones: Exploring zones whose environment characteristics are peculiar such as toilets, elevator access cavity or emergency stairway corridors, designated as irregular zones, lead to the total map breaks the symmetry, including by that a zone of differentiation where the robot easily makes its location.

3) Occasional return to previous positions: Another way to obtain a refined map that is by occasional return to previous positions, which allows the robot to upload the position on the reference map and overload the data acquired.

E. Hardware Arrangement

The arrangement way of the three components used to achieve the goal of mapping is represented in Fig. 2. The Kinect is represented by black rectangle on the image and the Lidar laser is represented by the blue shape over the robot.

This configuration is important because influence the values for the transformations reference frames of the Kinect depth camera and the Lidar laser. The Kinect depth sensor is align with the center of the robot, only changing the height in 28 cm from the */base_link*, given origin to the */camera_link* frame. The Lidar sensor is 4 cm ahead the Kinect and above the top of the robot, so the medium point is 22.5 cm higher the floor and 4 cm forward the */base_link* frame.

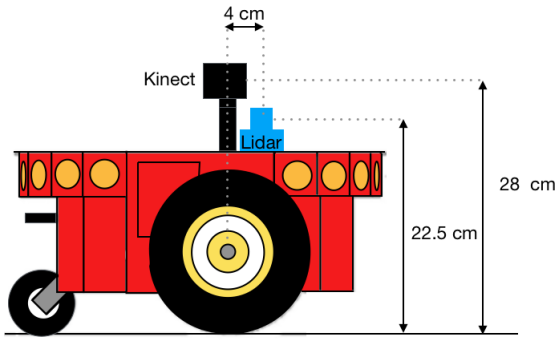

Fig. 2. Hardware Arrangement.

F. Occupancy Grid Algorithm

Initially, the sensor used to acquire information of the environment was a laser, because it has a simpler interpretation comparing with the camera, and the algorithm used was the one explained in the book [1]. However that algorithm is more focus on using the sonars instead of a laser or depth camera and, for that reason, was necessary to change for another approach. With the reformulation of the solution the algorithm used was the Bresenham Algorithm because of the simplicity for drawing lines in a grid composed of cells.

IV. Experimental Results

A. Simulations

According to the section III, which explains in detail the implementation, the first step passed by obtaining a map using *gmapping* for reference proposes, and the second step passed to implement the developed algorithm using Kinect depth camera sensor and the Pioneer robot. Although before applying the algorithm developed on the real system, there were made a group of simulations in python to analyze and evaluate our algorithm. Some of the experiences performed are represented in Fig. 3.

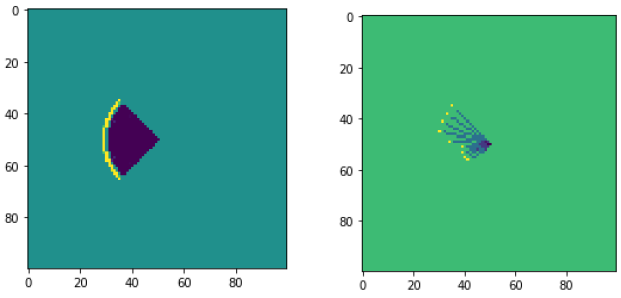

Fig. 3. Simulations in python with synthetic data.

In the left image, it was simulated synthetic data that detected an object in every beam and all measurement at the same distance. In this case, it was used 700 beams in order to get as close as possible the real situation, where the camera has many beams with minimal distances between each one. It is possible to observe that the program fills in the cells in the range of the sensor and with a distance smaller than the measurement as free, the cells that are at the distance of the measurement with a thickness tolerance as occupied and the rest of the map is unknown.

The right image is to show how is the algorithm working. In this situation, it was used just 10 beams equally distributed on the range of the sensor and with different measurements. As expected, the distance until the object found in the sensor range is free and in that distance is occupied.

B. Reference and obtained map of the 5th floor

The image of the Fig. 4 represents the result using ROS package gmapping with Laser Range Finder for the 5th floor and is used as reference map.

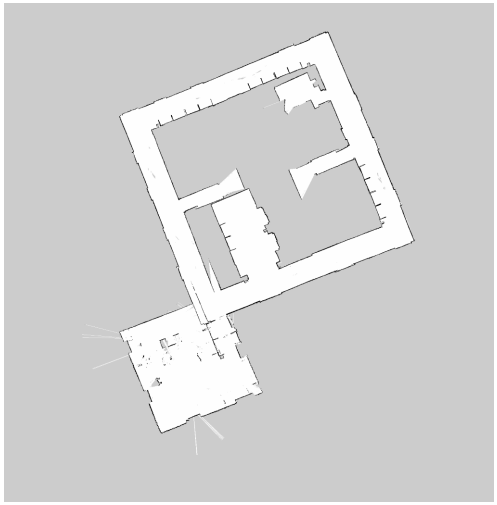

Fig. 4. Reference map of the 5th floor using ROS package gmapping [8] with Laser Range Finder.

The image of the Fig. 5 represents the result of the execution of the mapping algorithm developed for the 5th floor using Kinect depth camera sensor and the Pioneer robot.

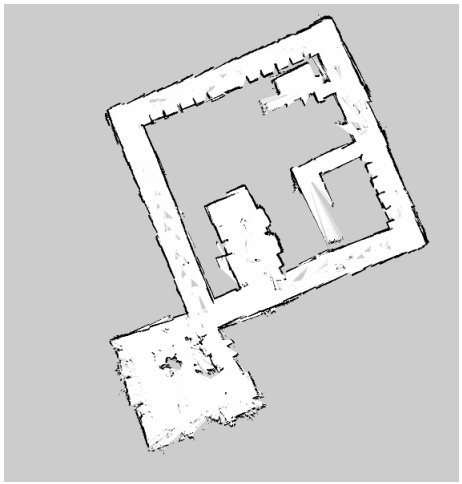

Fig. 5. Obtained map of the 5th floor using Kinect camera.

To evaluate how different the map obtained with the kinect sensor is in comparison to the gmapping reference, it was decided to first get a binary difference image (as in Fig. 6), where a difference occurrence of the occupation value in a

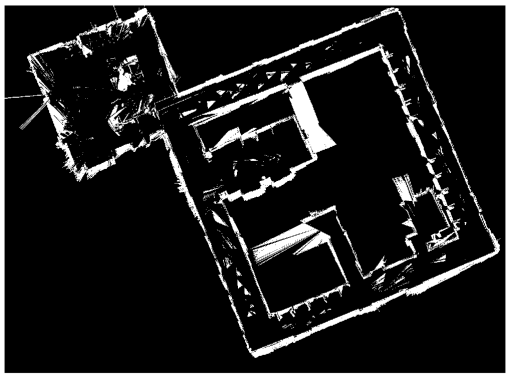

Fig. 6. Binary difference between the gmapping reference and the 5th floor map obtained with the depth camera.

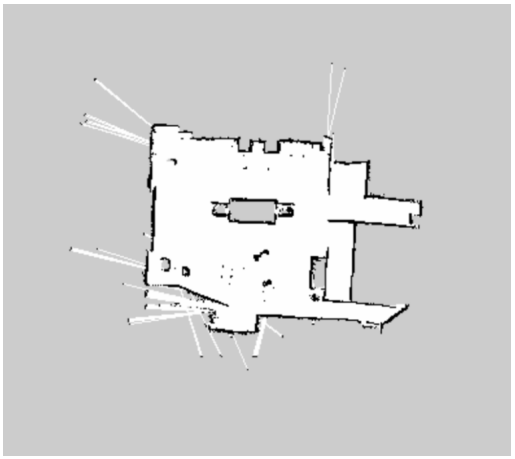

Fig. 7. Reference map of the 8th floor Testbed using ROS package gmapping [8] with Laser Range Finder.

certain point is marked as "1" (white), representing an error occurrence in that point in comparison to the obtained map. Likewise, points marked as "0" (black) represent the correct assignment of the occupation values.

That being said, the following equation is used as a quantitative analysis measurement of how similar the map is when compared to the reference,

$$\textbf{Error} = \frac{\sum_{i,j} B_{i,j}}{N_{occupied} + N_{free}} \times 100. \tag{7}$$

Where $B_{i,j}$ is the pixel values of the binary difference image (where "1" is an error and "0" is a correct assignment of the occupancy values), $N_{occupied}$ is the total number of occupied cells in the reference map and the N_{free} is the total number of free cells in the reference map.

As a result, in what concerns the mapping of the 5th floor, the binary difference image is represented in Fig. 6. Applying the error formula defined in equation (7), the result is an error of 26.2%.

In this case, the smaller perpendicular corridor represented on the left of the reference Fig. 4 is not mapped with the kinect sensor. This happened because the door to the stairs' corridor was closed during the mapping of the 5th floor with the kinect sensor. As a result, a white rectangle is represented in Fig. 6 in that position. This white rectangle was compensated in the error formula, as being, in fact, black (not an occupancy error).

C. Mapping of the 8th Floor Testbed

The map represented in Fig. 7 corresponds to the result of using the ROS package gmapping, with the Laser Rangefinder, in order to obtain a reference map for the 8th Floor Testbed environment.

The map in Fig. 8 represents the result of executing the developed mapping algorithm on the 8th Floor Testbed, with the Kinect depth camera and using the Pioneer robot.

Analogously to what has been done before, the binary difference image in this case is the one in Fig. 9. Applying the error formula defined in equation 7, the resulting error was of 30.42%.

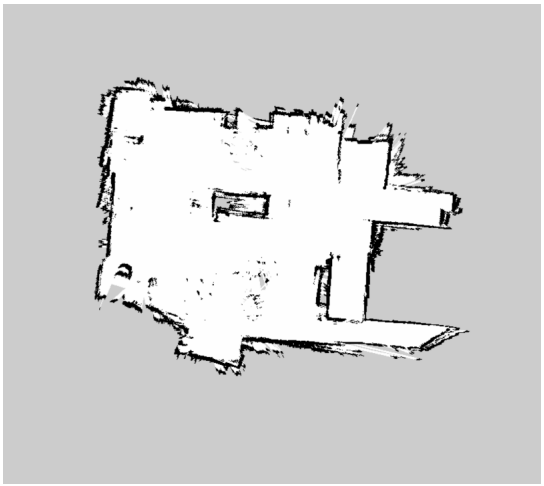

Fig. 8. Obtained map of the 8th floor Testbed using Kinect camera.

Fig. 9. Binary difference between the gmapping reference and the 8th floor testbed map obtained with the depth camera.

In this case, the mapped environment is larger, meaning, its places are less narrow like the corridors represented in Fig. 4.

D. Computational Time

It is essential to measure the computational time to evaluate the performance of the algorithm and prove its efficiency. The time each iteration of the algorithm takes was considered as a measurement of computational time and was computed through the Python `time` module [13]. The median of the time

of all iterations was considered. In Table I, it is possible to observe the time results obtained for simulation and different datasets. It is essential to mention that the obtained times did not take into account the time it takes to publish the map to ROS, as this time is strongly related to the resolution and dimensions of the map and is outside of our control. In our project, a better resolution map was considered more important than a low publishing map time, map with less resolution. Additionally, since the map is not dynamic, there was the need to allocate a sufficiently big initial map to ensure that the measurements would not be outside of it. If the time it takes to publish the map were considered, the times presented in Table I would increase by 15 seconds for a 3000 by 3000 map with 0.02 resolution.

TABLE I
COMPUTATIONAL TIME OF THE ALGORITHM FOR DIFFERENT DATASETS.

Dataset	Time [s]
Simulation	0.031
5th Floor	0.210
8th Floor - Testbed	0.320

The results presented in Table I were obtained using a 2.8GHz Dual-Core Intel Core i7 computer. The obtained results demonstrate that our algorithm is fully capable of running in real-time.

V. CONCLUSION

Taking into account all the results in the previous sections, the main goal of the project was accomplished, since the maps obtained with the developed algorithm are very similar while comparing with the ones that were used as reference maps.

However, there are many improvements that can be implemented. The error metric used, as mentioned in section IV, was the binary difference of a reference map and the one that our algorithm provides. The errors were not little, since there are many things that can influence that metric. One of the major problems is the use of a depth camera for data acquirement since the reference map was obtained with a laser, and we use the depthimage_to_laserscan package to convert a depth image into a laser scan. If the goal is to create a virtual laserscan from the RGBD device, and the sensor is forward-facing, depthimage_to_laserscan will be much more straightforward and efficient since it operates on image data instead of bulky pointclouds. However, if the sensor is angled, the node pointcloud_to_laserscan) may show to be very helpful [14].

Another error that should be taking into account is the fact that due to the low camera range, the laser can read a lot more information than the camera, and for that reason, even if they pass by the same path, they are going to capture different information (less information with the depth camera - increases mapping time and the complexity of the trajectory).

Furthermore, while publishing the map, our algorithm is slower (15 seconds slower) than when it is running without publishing (usually takes less than 0.5 seconds in every iteration), and that difference also affects our results.

Another thing that should be mentioned is the fact that while acquiring the map, the reference frame that is used is static, so when the algorithm is running, the map runs out this reference frame. One solution for that is the use of a dynamic reference frame that is built along with the map.

Finally, more experiences in other environments could have been made. However, due to the lack of material in the labs (for example there was only one charger for all the robots and they run out of battery very quickly) and because other groups want to use the robots as well, it was not possible to run any more tests.

REFERENCES

[1] S. Thrun, W. Burgard, D. Fox, and R.C. Arkin. *Probabilistic Robotics*. Intelligent Robotics and Autonomous Agents series. MIT Press, 2005.

[2] Colin Flanagan. The Bresenham Line-Drawing Algorithm. https://cs.helsinki.fi/group/goa/mallinnus/lines/bresenh.html. [Online; accessed 14-December-2019].

[3] Srećko Jurić-Kavelj. ROSARIA. http://wiki.ros.org/ROSARIA. [Online; accessed 14-December-2019].

[4] MobileRobots Inc. Pioneer 3 operations manual. http://mediawiki.isr.ist.utl.pt/images/0/00/P3OpMan3.pdf. [Online; accessed 14-December-2019].

[5] Mike O'Driscoll Chad Rockey. urg_node. http://wiki.ros.org/urg_node. [Online; accessed 14-December-2019].

[6] Graylin Trevor Jay. teleop_twist_keyboard. http://wiki.ros.org/teleop_twist_keyboard. [Online; accessed 14-December-2019].

[7] Brian Gerkey. AMCL. http://wiki.ros.org/amcl. [Online; accessed 14-December-2019].

[8] Brian Gerkey. gmapping. http://wiki.ros.org/gmapping. [Online; accessed 14-December-2019].

[9] Tony Pratkanis Brian Gerkey. map_server. http://wiki.ros.org/map_server. [Online; accessed 14-December-2019].

[10] Radu Bogdan Rusu Piyush Khandelwal Patrick Mihelich, Suat Gedikli. freenect_launch. http://wiki.ros.org/freenect_camera. [Online; accessed 14-December-2019].

[11] Chad Rockey. depthimage_to_laserscan. http://wiki.ros.org/depthimage_to_laserscan. [Online; accessed 14-December-2019].

[12] James Bowman Tim Field, Jeremy Leibs. rosbag. http://wiki.ros.org/rosbag. [Online; accessed 14-December-2019].

[13] Time access and conversions. https://docs.python.org/3/library/time.html. [Online; accessed 20-December-2019].

[14] Paul Bovbel. pointcloud_to_laserscan. http://wiki.ros.org/pointcloud_to_laserscan. [Online; accessed 22-December-2019].