Autonomous Systems

Occupancy Grid Mapping Using Microsoft Kinect

Project Report

Diogo Morgado – nº 84032, Luis Lopes – nº 84116

Abstract – In the current days robots play an important role in our society and more than ever it is important that humans and robots can coexist. So, it is imperative to a robot to be aware of its surroundings. A good way to do that is to have a map of the environment where the robot is.

Our objective with this work is to build a map of a certain environment using the Pioneer 3-DX robot and the Microsoft Kinect camera. To do that, a Occupancy Grid Mapping algorithm was implemented. This probabilistic algorithm divides the environment into small cells and attributes a probability to each one of them of being occupied or not.

The map was then compared to the one obtained with the ROS package GMapping in order to determine the quality of the results.

I. INTRODUCTION AND MOTIVATION

As humans, when navigating through new environments, we are capable of identifying objects, planning movements and characterizing environments using our natural senses and our brain's ability to fuse this information. However, since autonomous robots are not able to do all those things naturally, building a map is essential so that they are able to identify objects/obstacles and to understand and characterize the environment they are in. Also, the mapping problem is crucial in autonomous robots because it's used in activities like Motion Planning, Localization and Navigation.

In order to build an accurate map of the environment, reliable information about the robot pose is necessary. Since the localization problem is not part of our project, we started by using the robot odometry to get its current pose. However, since the odometry error accumulates over time, we tried to use the AMCL (*Adaptive Monte Carlo Localization)* package for ROS (*Robotic Operating System)* in order to get a more accurate pose of the robot. To do so, we also had to use the GMapping package for ROS because a map it's required to implement the AMCL algorithm.

II. METHODS AND ALGORITHMS

The main algorithm used in this project, to build a map of the environment, is the Occupancy Grid Mapping, which is a probabilistic mapping method that partitions the space to be mapped into finitely many grid cells with the same size and where each of those cells have an occupancy probability attached to it. The Occupancy Grid Mapping algorithm used is represented in Figure 1 [1] and it's described next.

A. Occupancy Grid Mapping Algorithm

For each iteration of this algorithm, the estimated occupancy probability of all the cells is updated, using the actual position and orientation of the robot, x_t, the ranges, z_t, measured with the distance sensors and the log-odds occupancy probability vector, $l_{t,i}$, which represents the occupancy probability of the cell *i* being occupied in instant *t*. It should be noticed that the probabilities are represented in the log-odds form to avoid

numerical instabilities for probabilities near 0 and 1.

```
1:  Algorithm occupancy_grid_mapping({l_{t-1,i}}, x_t, z_t):
2:      for all cells m_i do
3:          if m_i in perceptual field of z_t then
4:              l_{t,i} = l_{t-1,i} + inverse_sensor_model(m_i, x_t, z_t) - l_0
5:          else
6:              l_{t,i} = l_{t-1,i}
7:          endif
8:      endfor
9:      return {l_{t,i}}
```

Figure 1 - Occupancy Grid Mapping algorithm [1].

The Occupancy Grid Mapping algorithm, represented in Figure 1, starts by looping through all grid cells *i*, and it updates those that fall into the sensor cone of the measurement z_t. For those that fall into the sensor cone, it updates the occupancy probability value using the *Inverse_range_sensor_model* function, represented in Figure 2 [1], and the constant l_0, that represents the prior of occupancy represented in log-odds form. Otherwise, for the cells that don't fall into the sensor cone, the occupancy value will remain unchanged.

B. Inverse Range Sensor Model

This function is used to determine the occupancy value (in log-odds) which each cell, that falls into the sensor cone, should be updated with.

```
1:  Algorithm inverse_range_sensor_model(i, x_t, z_t):
2:      Let x_i, y_i be the center-of-mass of m_i
3:      r = sqrt((x_i - x)^2 + (y_i - y)^2)
4:      φ = atan2(y_i - y, x_i - x) - θ
5:      k = argmin_j |φ - θ_{j,sens}|
6:      if r > min(z_max, z_t^k + α/2) or |φ - θ_{k,sens}| > β/2 then
7:          return l_0
8:      if z_t^k < z_max and |r - z_max| < α/2
9:          return l_occ
10:     if r ≤ z_t^k
11:         return l_free
12:     endif
```

Figure 2 - Inverse Range Sensor Model algorithm [1].

This function starts by determining the range *r* for the center-of-mass of the cell *i* and the bearing ϕ of the center-of-mass in relation to the robot. After that, it determines the beam index *k* that is closer to the cell *i*. The range associated to the beam *k* is z_t^k.

After determining all these parameters, it starts by evaluating if the cell is outside the measurement range of the sensor beam *k* or if it lies more than $\frac{\alpha}{2}$ (where α represents the object thickness) behind the detected range z_t^k and, if so, the function will return the prior value of occupancy, l_0. Then, for the cases where the range measurement of the cell that is being evaluated is lower than the maximum range of the sensor, z_{max}, and within object thickness, the function will return l_{occ} ($l_{occ} > l_0$), which means that the cell will be marked as occupied. For the cases where the range measurement, z_t^k, is larger than the distance to the cell, *r*, the function will return l_{free} ($l_{free} < l_0$), which means that the cell will be marked as free.

C. Packages

Since we need to have all the range measurements to implement the Occupancy Grid Mapping algorithm, and since it's hard to work with *PointClouds* (which is the type of data that is received from the Microsoft Kinect depth camera), we decided to use the *Depthimage_to_laserscan[3]* package for ROS, which converts a depth image (from the Microsoft Kinect) to a laser scan.

Also, since the odometry error accumulates over time, we tried to use the AMCL[4] (Adaptive Monte Carlo Localization) package for ROS to try to get a better localization of the robot. However, since the AMCL package requires a map, we also had to use the GMapping[5] package for ROS.

III. IMPLEMENTATION

A. Occupancy Grid Mapping Algorithm

The algorithm, implemented in *Python*, is executed at a frequency of 0.46 Hz and, in each execution of the algorithm, the most recent range measurements and robot poses are used. Since the execution time of the algorithm increases with the size of the map, the map resolution depends on the size of the map that we are creating. For example, for a map of 30x30m a resolution of 0.3m was chose, so that the size of the map in pixels was 100x100, which is tolerable in terms of execution time. However, for smaller maps (for example 10x10m) we are able to choose a higher resolution (for example 0.1m). It should be noticed that, the higher is the resolution, the higher is the accuracy of the map and the execution time.

Since we assumed that the probability of a cell being occupied or free was equal, we defined, for the value of the prior for occupancy, l_0, the value 0.

B. Inverse Range Sensor Model

For the value of the object thickness, α, we considered it as 3/2 of the length of one cell of the map. Regarding to the value of the sensor cone aperture, β, we defined it as 0.001585 rad ($\simeq 0.091°$) because that's the angle increment between each laser beam of the laser scan obtained from the *Depthimage_to_laserscan* package. It should be noted that for values of β bigger than the defined, the algorithm would evaluate cells that would not have range measurement information.

For the occupancy values (in log-odds form) we chose the values 1 for l_{occ} and -1 for l_{free}. It should be noticed that, these values were chosen empirically, since these were the values that produced the most accurate results.

Also, for the maximum range value, z_{max}, we defined it as 6m, since that was the value we found to be the maximum range of the Microsoft Kinect.

C. Microsoft Kinect

The measurements used in the Occupancy Grid Mapping algorithm were taken using a Microsoft Kinect. This device has 3 types of sensors, one RGB camera, two 3D depth sensor and an array of microphones. For the purpose of this project only the two 3D depth sensors were used.

The Kinect's depth sensors produce a 640x480 pixels depth image with a horizontal field of view of 58º, a vertical field of view of 45º and an optimal range from 0.4 meters to 6 meters. The Kinect's depth sensors are able to provide a depth resolution of 1cm and a spatial *xy* resolution of 3mm at a range of 2 meters. The depth images are transmitted at a rate of 30 frames per second.

Once the camera stands on top of the robot while it moves, it was necessary to perform a static transformation between the *camera_link* and the robot's *base_link* in order to relate the two and correctly evaluate the occupancy of a certain location. The transformation applied is represented in Figure 3.

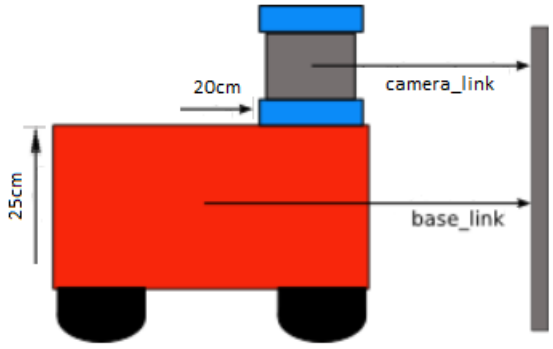

Figure 3 - Representation of the transformations applied.

D. Thresholding

In order to get the maximum likelihood map, the map built using our Occupancy Grid Mapping algorithm was submitted to a threshold process. To all cells that had occupancy probability over t_{occ}, it was attributed an occupancy probability of 1 and to all cells with occupancy probability under t_{free}, it was attributed an occupancy probability of 0. To all cells whose occupancy probability is between t_{occ} and t_{free}, it's attributed an occupancy probability of 0.5 (which means that its occupancy is unknown).

The values for t_{occ} and t_{free} that we chose were 0.75 and 0.25, respectively. We chose these values because, for the l_{occ} and l_{free} defined, the minimum value for a occupied cell (that was only evaluated one time as occupied) is 0.731, which corresponds to 1 (l_{occ}) in log-odds form and the maximum value for a free cell (that was only evaluated one time as free) is 0.269, which corresponds to -1 (l_{free}) in log-odds form. The result of the thresholding process can be observed in Figure 4.

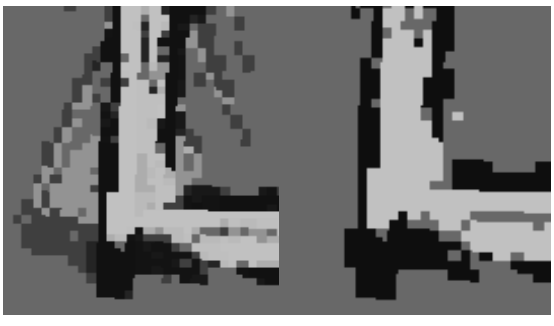

Figure 4 - Portion of the map of the 5th floor of the North tower before and after thresholding.

IV. EXPERIMENTAL RESULTS

In order to test our Occupancy Grid Mapping algorithm, the Pioneer-3DX, with the Microsoft Kinect on top, was teleoperated in different divisions of the North Tower. To test the robustness and accuracy of our algorithm, we also built different maps of the same division using different paths. The maps obtained, after and before the thresholding process, using our algorithm are represented in Figures 5-10.

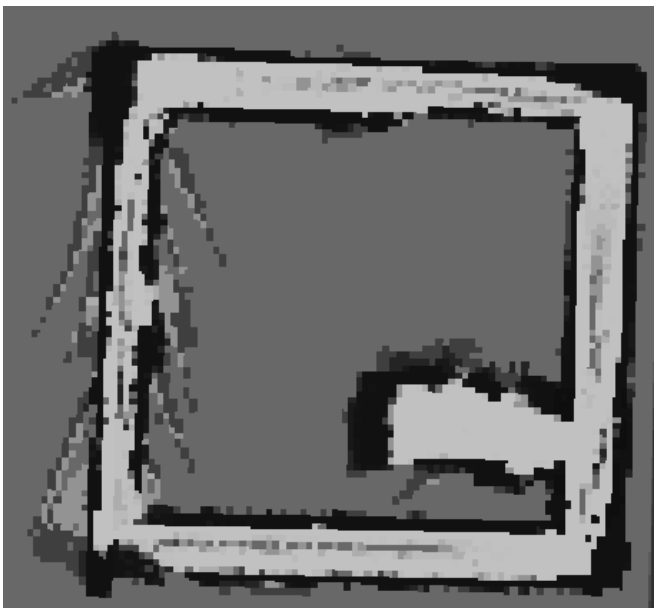

Figure 5 – Map of the 5th floor of the North Tower before thresholding.

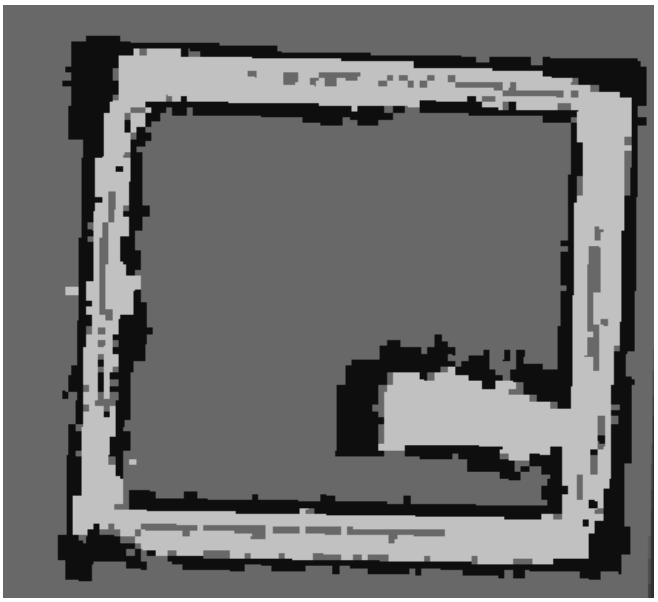

Figure 6 - Map of the 5th floor of the North Tower after thresholding.

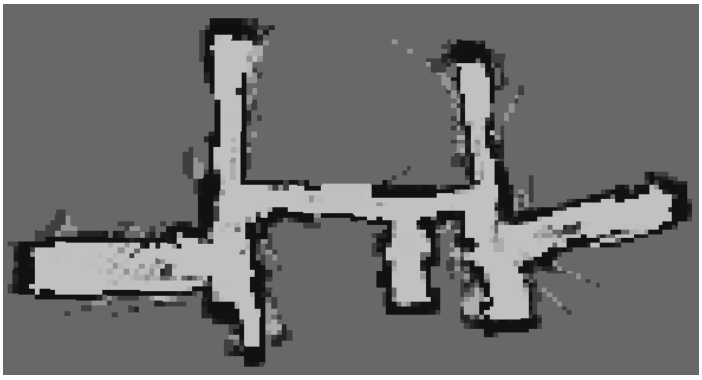

Figure 7 - Map of the 1st floor of the North Tower before thresholding.

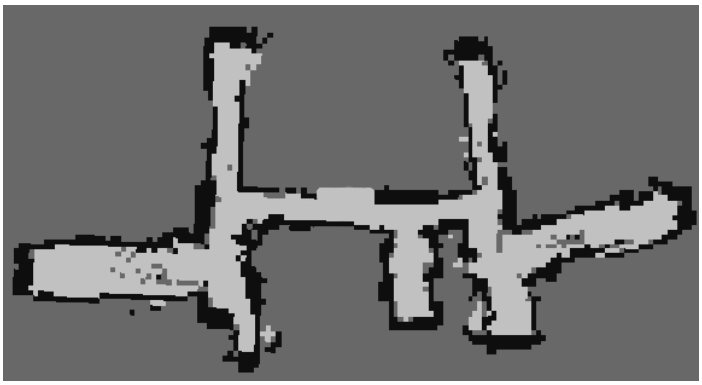

Figure 8 - Map of the 1st floor of the North Tower after thresholding.

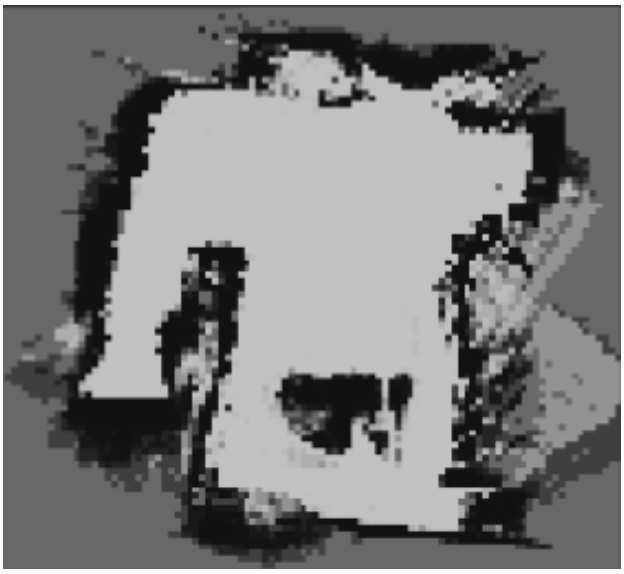

Figure 9 - Map of the LSDC4 classroom before thresholding.

Figure 10 - Map of the LSDC4 classroom after thresholding.

Observing the above figures, we can notice that after the thresholding process most of the outliers and noisy measurements are removed, that, beside some errors, the robot odometry seems to be a good way to have the robot pose at each time and that, at first sight, the maps looks accurate. However, since it's known that the odometry accumulates error over time, the AMCL package was used to try to get a more realistic robot pose. As said before, in order to use the AMCL algorithm we also had to use the Gmapping package to built a map. This being said, the map built using GMapping is represented in Figure 11 and the maps built (before and after the thresholding process) using our Occupancy Grid Mapping algorithm with the robot pose from AMCL are represented in Figures 12 and 13, respectively.

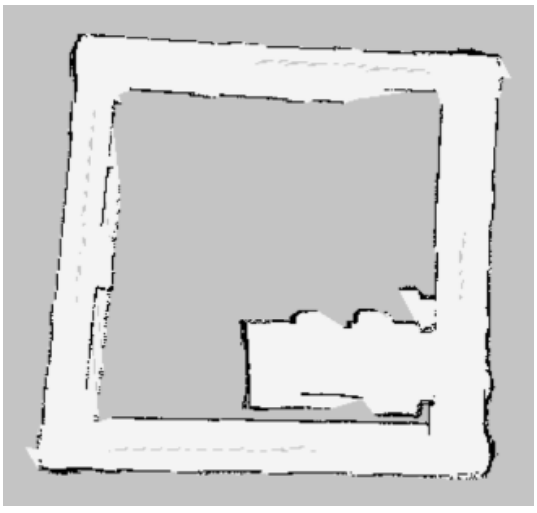

Figure 11 - Map of the 5th floor of the North Tower using GMapping.

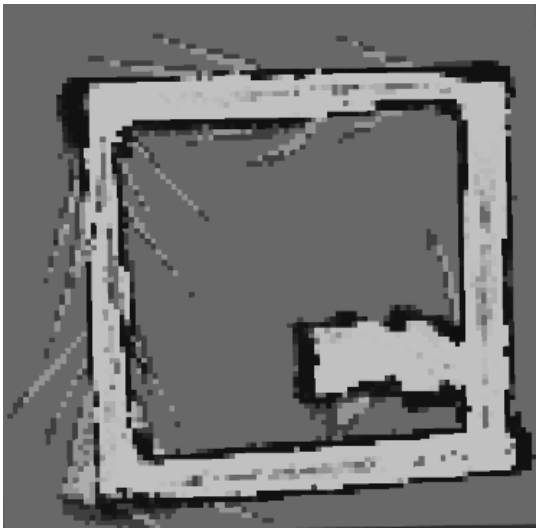

Figure 12 - Map of the 5th floor before thresholding and using AMCL.

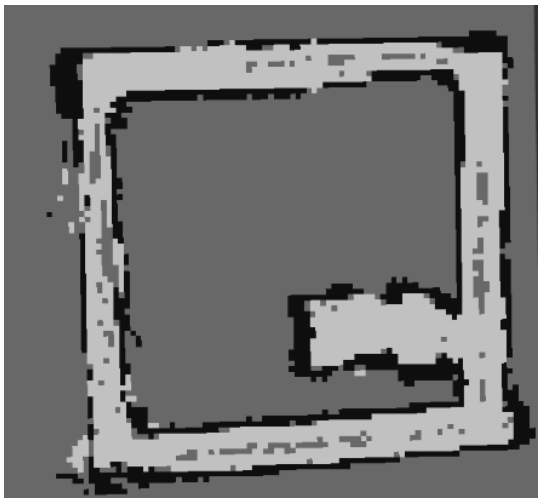

Figure 13 - Map of the 5th floor after thresholding and using AMCL.

Observing the maps represented in Figures 11, 12 and 13 and comparing them with the maps represented in Figures 5 and 6, we can notice that the results are similar.

Since GMapping package is able to produce a more accurate map, we are going to use the map represented in Figure 11 as a ground-truth map. In order to evaluate our map with respect to ground-truth, we compared the occupancy probability of all the cells of both maps (the map we built using our algorithm, represented in Figure 6 and the map built using GMapping, represented in Figure 11). The ratio of correct cells that we obtained was around 78%, which demonstrates that our algorithm is able to build accurate maps.

To understand if the map built using our algorithm with the pose from AMCL is better than the map built using our algorithm with the robot odometry, we also compared the occupancy probability of all cells of both maps, represented in Figures 11 and 13, and the ratio of correct cells that we obtained was around 75%, which is close to the ratio of correct cells of the map represented in Figure 6.

V. CONCLUSIONS

The maps obtained using only our algorithm and the odometry provided by the robot are, in most of cases, pretty decent when comparing to the same maps obtained using the package GMapping. Yet, in an attempt to improve the quality of our map, the package AMCL was used to get a more accurate robot localization. However, as it was shown previously, the results were very similar. This leads us to conclude that the odometry results obtained are very accurate.

The computational time of each iteration of our algorithm was also measured, where a value of 0.46 Hz was obtained. Comparing this value to the value obtained when using GMapping (0.36Hz) we can conclude that our Occupancy Grid Mapping algorithm is faster.

The imperfections of the maps can be associated to the fact that the *Depthimage_to_laserscan* package was used to convert the point cloud, coming from the Kinect depth sensor, to a laser scan. It's a known limitation of the depth sensors the zero readings when encountering reflections and the lack of readings when seeing trough glass. Also, the odometry used to get the robot localization, which is calculated based on the wheel motion, has an associated error which accumulates over time.

Overall it is safe to say that the algorithm implemented is stable and can estimate, with good quality, maps that can be then used by the robot to perform various functions.

VI. REFERENCES

[1] S. Thrun, W. Burgard and D. Fox, Probabilistic Robotics, 2005 MIT Press

[2] https://fenix.tecnico.ulisboa.pt/downloadFile/1689468335617258/07_MappingProbabilistic_1819.pdf

[3] http://wiki.ros.org/depthimage_to_laserscan

[4] http://wiki.ros.org/amcl

[5] http://wiki.ros.org/gmapping