Drone Mapping with Occupancy Grid Mapping

André Amaral (90012), Helder Pereira (90091),Nuno Veiga (90155), Pedro Pedrosa (90166)

***Abstract*— Maps are used to guide people through unknown places. Nowadays some tasks need to be done by robots so it is very important to be able to map an environment with a robot. Specially with drones, allowing us to control and monitor inaccessible places.**

The aim of this project is to develop an algorithm capable of mapping a 2D environment with precision using a drone. We used the Gazebo simulator with a PX4 drone. The algorithm we used was the Occupancy Grid Mapping and the Inverse Range Sensor Model.

To validate our results we compared our map with the one obtained with the ROS library - GMapping, using the Husky robot provided. Then, we used an image processing method in Matlab, so that we could extract some metrics between the maps we obtained and the ones from GMapping.

In addition to that, it was implemented a 3D variant of the algorithm, where we used the Occupancy Grid Mapping and the Inverse Range Sensor Model to map a three dimensional environment.

***Index Terms*— Occupancy Grid Mapping, Inverse Range Sensor, Bresenham's Algorithm, Mapping, PX4 Drone**

I. Introduction and Motivation

Humans have a huge set of qualities and one of them is to be able to map, in our brains, instantaneously, an environment. However Autonomous Robots do not have those abilities, so we need to teach them (using algorithms) how to map an environment with obstacles. Nowadays this problem is used in several activities such as, for example, in spacial exploration.

The purpose of this paper is to build an accurate map in 2D, using a LiDAR sensor, assuming we know the robot pose. Different environments will be mapped in order to test the robustness of our algorithm.

II. Methods and Algorithms

Before we start defining algorithms, we needed to take in consideration some assumptions such as the fact that the occupancy is binary, the world is static, cells are independent and the model is perfect. After that, to build a map, the first algorithm to take in consideration is the *Occupancy Grid Mapping*.

To implement this algorithm we need to introduce another one, the *Inverse Range Sensor Model*. This second algorithm implements the inverse measurements in log-odd form and assigns values that correspond to the cell state (free, occupied and unknown).

In order to improve the computational velocity of that algorithm, we did some research and we found that was possible with an already established one, the *Bresenham's Algorithm*. This third algorithm allows us to apply the Inverse Range Sensor Model only to the cells that are in the path of the laser beam.

A. Occupancy Grid Mapping

This probabilistic algorithm divides space into grid cells of the same size, each cell has a number that represents the state of occupancy, which will be updated after each measurement.

Special thanks to professor Rui Bettencourt for helping us with the GMapping problem we had with the Husky robot. It really was a phenomenal help for the development of the project. We also want to say thanks to professor Rodrigo Ventura for all the constructive comments in each middle presentation.

For each position of the robot, the cells, in the range of the LiDAR sensor beam, will be analysed. If the information, after a measurement, leads to an update, the cell number will change to the previous value plus the information received from the *Inverse Range Sensor Model*.

In figure 1 we can see the pseudo code which was the role model for our implementation of this algorithm.

Algorithm of occupancy grid mapping($\{l_{t-1,i}; i\}$; x_t ; z_t):
for time from 1 to T do
 for all cells m_i do
  if m_i in perceptual field of z_t then
   $l_{t,i} = l_{t-1,i}$ + inverse sensor model(m_i; x_t ; z_t) + l_0
  else
   $l_{t,i} = l_{t-1,i}$
  end if
 end for

Fig. 1. *Occupancy Grid Mapping Algorithm* pseudo code

B. Log Odds Function

This conversion function, although simple, is very important, because the *Inverse Range Sensor Model* will return logarithmic values and in the *Occupancy Grid Mapping* we will need probabilistic values.

In the equation 1 we can see how to obtain a probabilistic value from a logarithmic one:

$$P(m|z_{1:t}, x_{1:t}) = 1 - \frac{1}{1 + e^{l_{t,i}}} \tag{1}$$

C. Inverse Range Sensor Model

As mentioned before this algorithm will define the value of each cell.

To do that we need, on a first approach, to calculate the distance between the robot and the cell, the angle range which is defined as the angle between adjacent beams to the one we are analysing.

Now we have 3 options:

Unknown Case) if the target cell is either out of the range or out of the angles range, it returns the value of occupancy that was previously in the cell, so there is no modification in the value.

Occupied Case) if the target cell is in range and near the boundaries of the obstacle it returns an occupied state (l_{occ} = 1)

Free Case) if the target cell is closer to the robot than the obstacle, the cell is free and it returns a free state (l_{free} = -1)

In figure 2 we can see the pseudo code which was the role model for our implementation of this algorithm.

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

Fig. 2. *Inverse Range Sensor Model* pseudo code

D. Bresenham's Algorithm

Although this algorithm has not been addressed in the subject, we implemented it in order to reduce the computational complexity. For each LiDAR laser beam, the algorithm finds each cell intersected by the beam and the *Inverse Range Sensor Model* will only be applied to those cells. With this algorithm the velocity of the *Inverse Range Sensor Model* is greatly increased. Consequently, the parameter β in figure 2 becomes obsolete. The used version of the Bresenham's algorithm is modified to process line segments with all possible slopes and directions, as shown in the pseudo-code in figure 3.

```
line_start = (x1,y1);line_end = (x2,y2);
Modified_Bresenham_algor((x1,y1),(x2,y2)):
   if (|line_slope|>1):
      is_steep = 1;
      exchange(x1,y1);
      exchange(x2,y2);
   swapped = false;
   if (x1 > x2):
      exchange(x1,x2);
      exchange(y1,y2);
      swapped = true;
   dx = x2 - x1;
   dy = y2 - y1;
   error = integer part of (dx / 2.0);
   ystep = 1 if y1 < y2 else -1;
   y = y1;
   point_list = [];
   for x in range(x1, x2 + 1):
       coord = (y, x) if is_steep else (x, y);
       attach coord to point_list;
       decrease |dy| from error;
       if(error < 0):
          increase y by ystep;
          increase error by dx;
   if (swapped) return point_list from the end
   else return point_list from the beginning
```

Fig. 3. *Bresenham's Algorithm* pseudo code

III. Implementation

A. Algorithm Input And Output

In order to obtain a map, the information needed is the drone's position, which we obtained directly from the simulator, since it outputs the position ground truth, and also a range sensor. For the range sensor, we opted for a 2D *LiDAR* with 360º vision and 640 beams.

It is also key to define the dimensions of the 2D map. The values of height and width of the map can be defined in code, given the dimensions of the environment to be mapped, they are not auto-adjusted.

The default map resolution is 0.05 and the default height and width are 500 cells and 1000 cells respectively. The map origin is dependent on the maps we are trying to analyze and apply our mapping method to, and for the tests presented for this report, the map origin corresponds to $x = -35$ and $y = -13$.

The algorithm outputs a 2D map as well as a 3D map, and the drone trajectory during the mapping process. In figure 4 we can see in detail the inputs and the outputs of the algorithm.

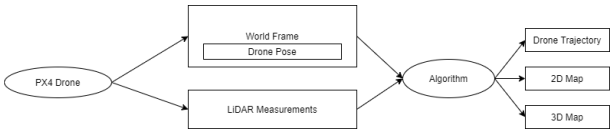

Fig. 4. *Algorithm graph representation*

B. Inverse range Sensor Model

This algorithm was implemented so that we could determine the state of each cell.

In order to do that, we had to define multiple parameters that would allow us to reach our goal. This was done by analyzing the respective topic that was subscribed.

First, we set the maximum number of sensor beams as $\theta = 640$, the maximum range of our sensor as $z = 10$ meters, the thickness of the obstacle as $\alpha = 0.2$ and the angular width as the the double value of the angle increment, therefore, $\beta \approx 0.0196$.

C. Occupancy Grid Map

While testing our algorithm, we realized that by moving the drone, the roll and pitch values would interfere in the results as the sensor wouldn't be horizontal. We decided to fix this issue by limiting the moment when the sensor reads values for when the roll and pitch values are lower than 0.05. The Inverse Range Sensor Model adds those values to a matrix that stores all the updated values.

Since the Inverse Range Sensor Model returns logarithmic values we use the Log-Odd Function to convert those values to probabilities. Finally we convert the matrix to a 1D array so that each value is assigned to the map and consequentially the cells are updated.

In order to be able to visualize the map we published a new topic which contains the map made by our algorithm. Using the *Rviz* we were able to visualise the map obtained with our algorithm.

IV. Experimental Results

A. 2D Mapping

Using *Gazebo* simulator, the *Rviz* and taking in consideration the implementation defined before, we began to run tests of our code using pre-recorded *rosbags*.

We chose 3 environments to map which will be presented in the following subsections.

1) First Tested Environment: The first environment we set up was the one present in figure 5.

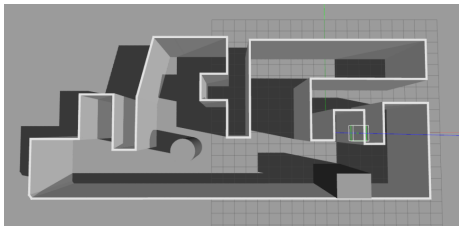

Fig. 5. *Gazebo Environment Number 1*

Applying our code we obtained the following 2D mapping among with the drone trajectory, present in figure 6.

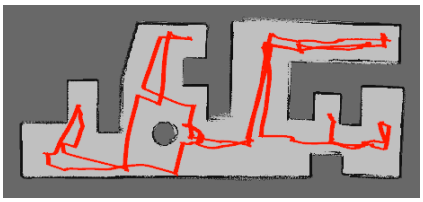

Fig. 6. *2D Mapping Number 1*

Applying the *GMapping* code, which is the best known code for this purpose, we obtained the following mapping, present in figure 7.

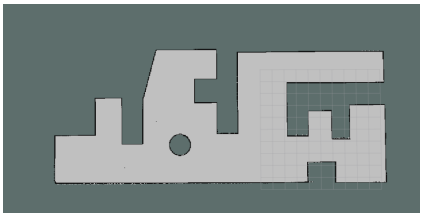

Fig. 7. *2D GMapping Number 1*

Even though visually the results speak for themselves, we introduced some comparative metrics. The following metrics were obtained from an image processing method made using Matlab.

This map had a Mean Square Error of 8.96% compared to the one obtained with *Gmapping*.

Our map has 62.37% of free area and the *Gmapping* 66.42%. Adding to these metrics, we also obtained the difference between the two images as we can see in figure 8.

Fig. 8. *Image Difference Simulation 1*

2) Second Tested Environment: The second environment we set up was the one present in figure 9.

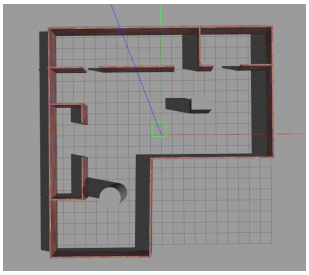

Fig. 9. *Gazebo Environment Number 2*

Applying our code we obtained the following 2D mapping among with the drone trajectory, present in figure 10.

Applying the *GMapping* code, which is the best known code for this purpose, we obtained the following mapping, present in figure 11

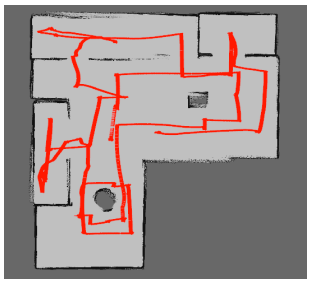

Fig. 10. *2D Mapping Number 2*

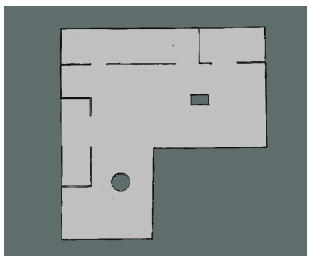

Fig. 11. *2D GMapping Number 2*

As we did for the previous simulation, we obtained comparative metrics. This map had a Mean Square Error of 7.64% compared to the one obtained with *Gmapping*. Our map has 70.88% of free area and the *Gmapping* 75.51%. Adding to these metrics, we also obtained the difference between the two images as we can see in figure 12.

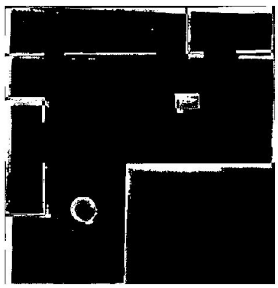

Fig. 12. *Image Difference Simulation 2*

3) Third Tested Environment: To prove the consistency of our algorithm we will map one last environment. As a third and last environment, we set up the environment present in figure 13

Applying our code, as we did in the previous simulations we obtained the following 2D mapping among with the drone trajectory, present in figure 14

Applying the *GMapping* code we obtained the map present in figure 15

As we did for the previous simulations we obtained comparative metrics. This map had a Mean Square Error of 5.51% compared to the one obtained with *Gmapping*. Our map has 75.28% of free area and the *Gmapping* 77.54%. The difference between the two images can be seen in figure 16.

When we analyze the results, the Mean Square Error for the three map simulations is different from zero (as expected) since our mapping is not as perfect as the one from Gmapping, even though it is low. The same happens in the free area where the values are almost identical. The difference, in the free area, between the two algorithms can be visualized in the respective image difference. In

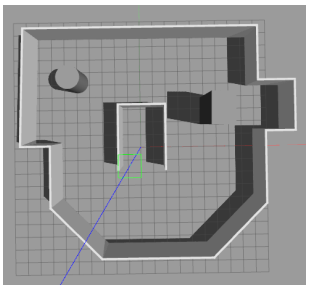

Fig. 13. *Gazebo Environment Number 3*

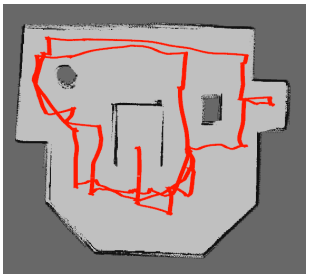

Fig. 14. *2D Mapping Number 3*

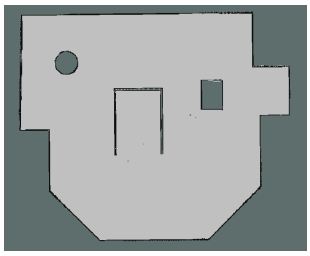

Fig. 15. *2D Mapping Number 3*

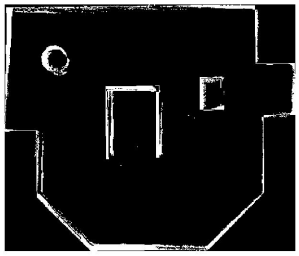

Fig. 16. *Image Difference Simulation 3*

an overall quantitative analysis, our algorithm was able to obtain results similar but not as perfect as Gmapping.

Finally, doing a qualitative analysis, by observing the resultant maps of both algorithms we can see some differences, but overall the environment is well mapped in both situations. The main differences can be found in some walls that are a little bit blurry and not as well defined as they could be.

One of the reasons behind that may be some desynchronization between the drone pose and LiDAR laser beams information, as they are approximately synchronized, but not exactly. Also, as the drone moves through the map it can have undesired inclinations that can lead to bad measurements.

As said before, the algorithm filters out information if the drone surpasses a defined value of pitch and roll, but it lets through small variations so that the filtering is not too harsh. This can lead to measurements that are not accurate, resulting in some errors in the final map.

By testing in several computers, we concluded that the computing power of the machine used has a large impact on the results. For a better computer, not only the algorithm is faster but also the control of the drone movement is smoother, which impacts the final results in a positive way.

B. 3D Mapping

Finally, we made a 3D Map, reusing our 2D code. To do that we defined a new environment which is more interesting to make a 3D Mapping, due to the relevant object forms that help obtain clearer conclusions. We did not have a comparison algorithm like we have in the 2D mapping, but if we look to the environment and compare it with our mapping we can conclude that they match, even though there's lots of room for improvement.

In figure 17 we have the environment and in figure 18 we have the 3D mapping among with the drone trajectory. The 3D map consists of a 3D grid. Each horizontal layer is an independent map. The applied algorithm used is exactly the same as in the 2D case, but now it is only applied to the layer where the drone is.

As a first observation, this approach will inherit any issue or improvement that the 2D mapping has. Secondly, since it uses 3D grid, big maps and small resolutions will take a lot of memory. One possible solution might be using a octree or use a quadtree for each height.

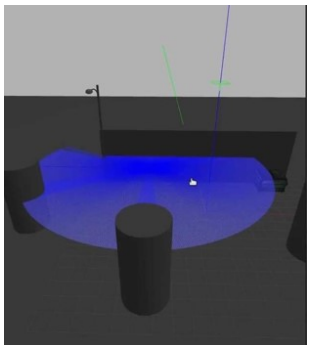

Fig. 17. *Gazebo Environment*

Applying our code, as we did in the previous simulations we obtained the 3D map among with the drone trajectory, present in figure 18.

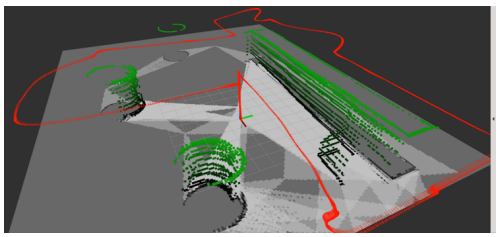

Fig. 18. *3D Mapping*

V. Conclusions

As we know the learning curve is steep in these kinds of projects, and the group had to overcome several issues along the way, that included setting up the simulation environment, synchronization issues, performance issues, among others. Also, mapping an environment

with a drone and a 2D LiDAR is challenging, as it has an inconstant movement that can lead to many measurement errors. Step by step these issues were solved and resulted in a good algorithm that produces good results, with some room for improvement. Overall, our algorithm was able to produce accurate maps. When compared to a top-tier algorithm such as Gmapping, the results were satisfying, but not perfect, which was expected.

Some improvements can be made to the algorithm to improve the results. In order to better handle drone pitch and roll changes, a way of using orientation values to obtain the relative 2D distances measured by the drone would reduce the filtering of information that is made in the algorithm and further improve the results. Also, if the position and *LiDAR* information came completely synchronized, the resulting map would be much more defined and accurate. So, a way of obtaining exactly synchronized information would also improve the results.

Finally, we were able to prove that the developed algorithm is easily translated to 3D, by developing our 3D mapping algorithm that replicates the initial mapping through the z axis. This algorithm produced good results, as shown previously.

REFERENCES

[1] The Bresenham Line-Drawing Algorithm. Colin Flanagan. [Online]. Available: https://www.cs.helsinki.fi/group/goa/mallinnus/lines/bresenh.html.

[2] Abrash, Michael (1997). Michael Abrash's graphics programming black book. Chapter 35: Bresenham Is Fast and Fast Is Cool [Online]. Available: http://www.phatcode.net/res/224/files/html/ch35/35-01.html.

[3] Robotic Mapping - Occupancy Grid Map. [Online]. Available: https://www.youtube.com/watch?v=Ko7SWZQIawM.

[4] Occupancy Grid Maps (Cyrill Stachniss, 2020). [Online]. Available: https://www.youtube.com/watch?v=v-Rm9TUG9LA&t=2585s.

[5] Hadji S.E, Hing T.H, Ali M.S.M, Khattak M.A and Kazi S; 2D occupancy grid mapping with inverse range sensor model, Asian Control Conference (ASCC), September 2015.

[6] S. Thrun, W. Burgard e D. Fox, 'Probabilistic Robotics', 2005 MIT Press