Fast Occupancy Grid Generation (FOGG) Using a RGB-D Sensor

Alex Wallar
Computer Science and Artificial Intelligence Laboratory
Massachusetts Institute of Technology
wallar@mit.edu

Abstract

In the domain of mobile robotics, the ability to determine the locations of obstacles is crucial. Many robotic platforms have access to RGB-D cameras which are able to determine the relative position of objects, however topological planning and collision avoidance algorithms rely on a two dimensional representation of the environment known as an occupancy grid. In this paper, I describe an algorithm that is able to generate an occupancy grid from point cloud data gathered from a Microsoft Kinect. While many implementations already exists, I show that this algorithm, FOGG, is able to run at the same rate in which data is being provided by the Kinect. The approach is also able to extrapolate geometric information about obstacles from the point cloud to determine an two dimensional projection of the environment given a single frame.

1. Introduction

Being able to move a robot from an initial configuration to a goal configuration without coming into collision with static and dynamic obstacles has been a major area of research [5] in mobile robotics. In order to move through the environment, a robot needs to be able to determine the positions and shapes of obstacles around it. Many robotic platforms are equipped with RGB-D sensors that are able to determine the relative positions of objects with respect to the robot. These sensors, such as stereo cameras and the Microsoft Kinect are able to provide 3D point cloud information, however topological planning algorithms, such as [7], rely on a two dimensional representation of the environment such as an occupancy grid. In this paper, I present an algorithm that is able to generate an occupancy grid using point cloud data gathered from a Microsoft Kinect. The algorithm is able to generate occupancy grids at the same rate in which data is being provided by the Kinect (30 Hz) and is able to extrapolate information about the obstacles' shapes from a single frame.

Whilst many similar implementations exist for converting a point cloud into a two dimensional occupancy grid, they require many frames in order to construct an accurate representation of the environment and therefore take longer to provide spatial information of the obstacles. The method I proposed, FOGG, is runs at 30 Hz and is able to extrapolate spatial information of the obstacles from a single frame.

The motivation behind this work is fast collision avoidance using topological planners. Since these planners act reactively, the obstacle geometric information does not need to perfectly accurate, but needs be very fast. That is why throughout the paper, assumptions are made which allow the algorithm run at a much higher speed, at the cost of accuracy. However, even though the algorithm may not be incredibly accurate in terms of the shapes of the obstacles, it will never indicate that a space is free when it is actually occupied by an obstacle.

2. Related Work

In terms of topological planning, the current state of the art for generating occupancy grids in real time for collision avoidance has been done by Lau et. al. which uses a single beam laser range finder to predict the motions of groups of humans, to extrapolate, and to generate an occupancy grid for for topological planning [8]. In terms of generating an occupancy grid from sensor information, variants of the simultaneous localization and mapping algorithm (SLAM) have made great strides into obtaining spatial information about the environment from on board sensors [4, 9, 2]. These approaches use on board sensor information as well as visual sensors to obtain relative position information for a robot. However, these approaches need multiple frames to be able to determine the geometry of obstacles. Also these works are more suited for localization rather than collision avoidance and thus are not suitable for the goal in mind which is topological collision avoidance.

The most prominent and advances SLAM algorithm using an RGB-D sensor is RGB-D SLAM V2 [3] which builds an incremental octomap using a Kinect sensor. Again, this approach is more suited for mapping and localization rather than collision avoidance and fast obstacle detection.

There have also been some people on the internet using a naive approach which is simply projecting the point cloud down to 2D which may seem like a valid approach if speed is the main concern. However, simply projecting the point cloud down to 2D does not let you extrapolate information about the obstacles' geometry and thus, to get a complete picture of the environment, multiple frames would be needed.

3. Approach

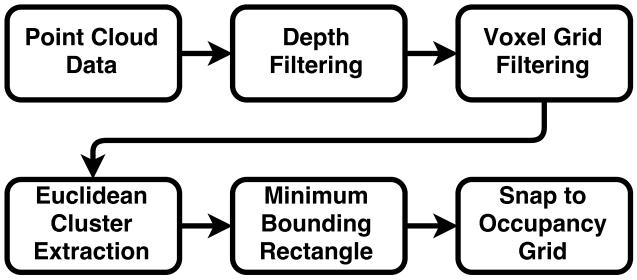

Figure 1. Diagram showing how FOGG processes the point cloud to produce an occupancy grid

3.1. Overview

A five step process is used by FOGG to generate an occupancy grid from point cloud data. First, the raw point cloud is depth filtered to remove all points that lie to far away from the robot to matter. The depth filtered point cloud is then passed to a Voxel filter [6] which downsamples the point cloud to reduce the number of points. Once the size of the point cloud is reduced, Euclidean cluster extraction is performed to segment the point cloud into multiple connected components whose points are at most a certain distance away from each other. A minimum bounding rectangle is then fitted to each cluster. To generate the resulting occupancy grid, a grid is instantiated, and if a point lies in any of the rectangles, that point is labelled as occupied, otherwise it is labelled as empty. The process proposed by the FOGG algorithm is shown in Fig. 1. Snapshots of the process at different stages is shown in Fig. 2.

3.2. Depth Filtering

In order to generate an occupancy grid given point cloud data from an RGB-D sensor, we first need depth filter the point cloud. This means that we get rid of all points whose depth is not within a given interval. This interval will vary between use cases, but for my experiments all points whose depths were larger than 1.8 meters were discarded. This allows the algorithm to focus on obstacles that the robot is more likely to collide with. Also, depth filtering reduces the over all size of the point cloud which allows the segmentation to run at a much faster rate.

3.3. Voxel Grid Filtering

Once the raw point cloud has been filtered based on the depth, a Voxel grid filter is applied. This also heavily reduces the size of the point cloud by guaranteeing that any two points in the resultant cloud will be at least a given distance away from each other. A Voxel grid filter allows the point cloud to be downsampled. This is important when using many RGB-D sensors such as the Kinect because they are usually highly dense, with a lot of redundant information. By applying a Voxel grid filter, on the already depth limited point cloud, a speed up from 5 Hz to 30 Hz is observed for generating the occupancy grid.

3.4. Euclidean Cluster Extraction

In order to place a bounding rectangle around each obstacle, we must first know which points belong to which obstacle. I employ Euclidean Cluster Extraction to cluster the points that belong to each obstacle together. This works by clustering points such that for each cluster there exists two points that are at most a given distance away from each other. Unlike segmentation for a monocular image, the raw point cloud provides Euclidean points in 3D space. This allows us to use this out of the box extraction algorithm provided by the Point Cloud Library [11].

3.5. Minimum Bounding Rectangle

Once the filtered point cloud has been partitioned into clusters, we can place a minimum bounding rectangle around each cluster. This rectangle allows us to extrapolate information about the obstacle geometry that we cannot see with a single frame (i.ethe back of the obstacle). To determine the parameters of the rectangle, I project each cluster to 2D. One possible problem with this is that parts of the obstacles that do not lie on the ground plane will be represented in the minimum bounding rectangle. I initially fit a cylinder on to each cluster, however this slowed down the algorithm from 30 Hz to 20 Hz. I employ the minimum area rectangle algorithm provided by the OpenCV Library [1] to determine the minimum bounding rectangle.

3.6. Occupancy Grid Generation

To generate the final occupancy grid, I instantiate a boolean grid with a user chosen resolution and origin. I then iterate through the cells in the grid, I convert the cell into Cartesian coordinates, and check whether or not that point lies in any of the rectangles. If the point lies in a rectangle, the cell gets marked as occupied otherwise the cell is marked as free.

3.7. Implementation

Four main libraries were used in the implementation to make ensure that the algorithms were vetted and as efficient

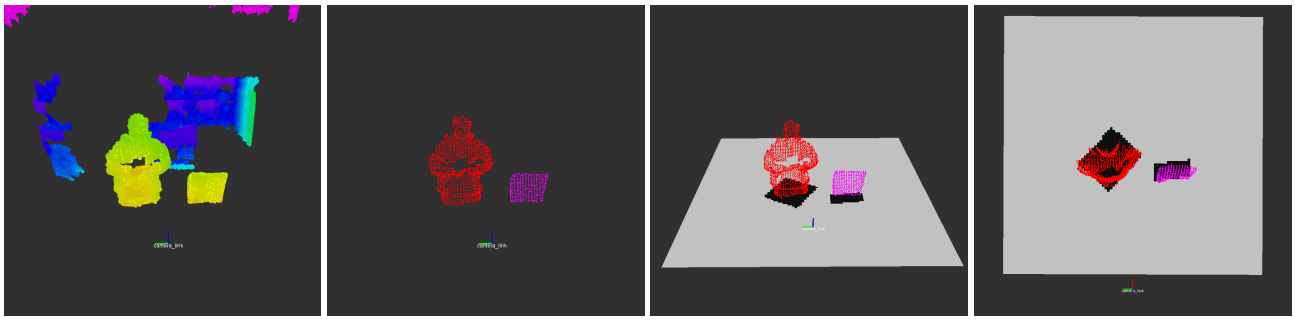

Figure 2. Images describing the iterative process of generating an occupancy grid from a point cloud. The first image is the raw point cloud obtained from the Kinect. The second image shows the depth filtered, Voxel filtered, and segmented point cloud. The third image shows the segmented point cloud along with the generated occupancy grid. Lastly, the fourth image shows a vertical perspective of the third image which allows you to see the extrapolated shape of the object generated by the FOGG algorithm. In image one, the color represents the depth. In images two through four, the color represents the cluster to which a point belongs.

as possible. The Freenect library was used to obtain the raw stream from the Kinect. ROS was used to take this stream and create a point cloud. A ROS node was written to run the FOGG algorithm. This ROS node subscribes to a point cloud topic and uses PCL and OpenCV to run the steps shown in Fig. 1. Once the occupancy grid has been generated, the FOGG ROS node publishes an Occuapncy-Grid ROS message as well as a PointCloud ROS message containing the labelled segmented point cloud. To visualize the labelled point clouds and the occupancy grids, ROS' RViz was used. This process is show in Fig. 3. The code I produced has been made open source and is hosted on GitHub [12].

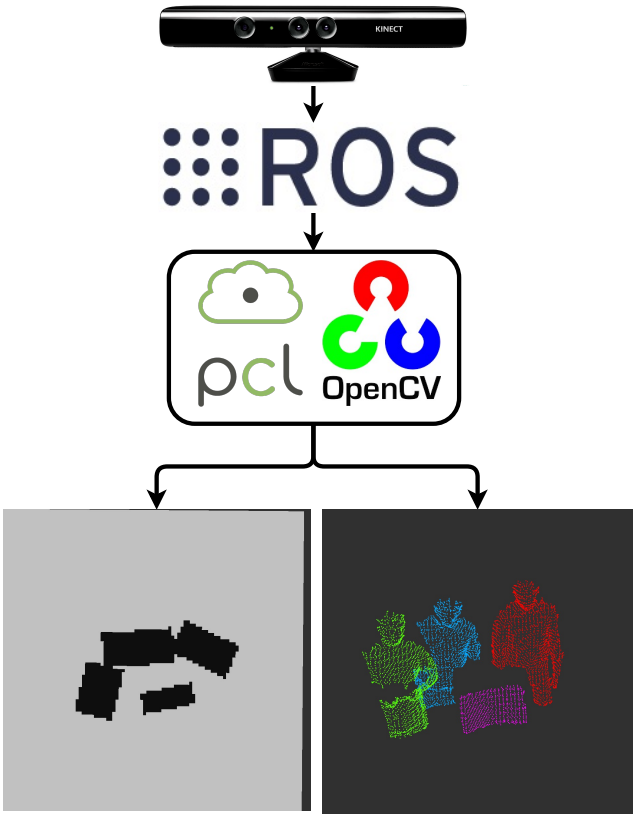

Figure 3. Diagram showing the data flow for FOGG. First a raw point cloud is obtained from the Kinect using the Robot Operating System (ROS) [10]. The point cloud is then processed by FOGG using the Point Cloud Library (PCL) [11] and OpenCV [1]. The resulting occupancy grid and segmented point cloud are then published to designated topics using ROS.

4. Results

The major contribution of the work is the speed. FOGG is able to produce occupancy grids as fast as it is receiving data from the Kinect (30 Hz). This is made possible by the heavy use of filtering in order to reduce the density and size of the point cloud. In terms of obstacle detection, the algorithm is at the mercy of the sensor. If the sensor is not able to detect an object (as is the problem with the Kinect if an object is too close to it), then FOGG will also not be able to generate a bounding rectangle around the object. In terms of accuracy, FOGG will always be overly cautious in terms of the obstacle geometry and indicate points which are not part of the obstacle as occupied in the final occupancy grid. However, this is the cost of being able to run the algorithm at 30 Hz. Also, these "over predictions" are errors in the extrapolation of the obstacle geometry and can be minimized with a more sophisticated method of extrapolation. The developed algorithm as it currently stands, is sufficient for future research I would like to conduct with topological collision avoidance and planning.

5. Future Work

In the future, I would like to use a deep learning method such as a neural network that would be able to take a point cloud cluster and return a parametric model that would let

me better extrapolate what the Kinect cannot see. I could train a neural network on many different point clouds from the Kinect to learn different parametric models such as car or person, then I could use the actual point cloud to determine the parameters the model.

I would also like to apply FOGG to the topological planning and collision avoidance algorithms that I have been working on and test the algorithms on a iRobot Create and a quadrotor.

6. Conclusion

This paper proposes a method for fast occupancy grid generation (FOGG) using a RGB-D sensor. This is done by first filtering the raw point cloud to reduce its density and size. Clusters are then extracted from the filtered point cloud and a minimum bounding rectangle is fitted for each cluster. These rectangles are then added to the occupancy grid. This method has been shown to run in real time and as fast as point clouds are being generated by the Microsoft Kinect. This method could be improved by adding more sophisticated extrapolation methods or deep learning neural networks as described in Sec. 5. However, FOGG at its current state is sufficient to begin testing topological planning methods on mobile robots.

References

[1] G. Bradski. The opencv library. *Dr. Dobb's Journal of Software Tools*, 2000.

[2] H. Choset and K. Nagatani. Topological simultaneous localization and mapping (slam): toward exact localization without explicit localization. *Robotics and Automation, IEEE Transactions on*, 17(2):125–137, Apr 2001.

[3] F. Endres. Rgbdslam v2. `http://felixendres.github.io/rgbdslam_v2/`. Accessed: 2015-12-12.

[4] J. Engel, T. Schöps, and D. Cremers. Lsd-slam: Large-scale direct monocular slam. In *Computer Vision–ECCV 2014*, pages 834–849. Springer, 2014.

[5] C. Goerzen, Z. Kong, and B. Mettler. A survey of motion planning algorithms from the perspective of autonomous uav guidance. *Journal of Intelligent and Robotic Systems*, 57(1-4):65–100, 2010.

[6] J. F. Hughes, A. Van Dam, J. D. Foley, and S. K. Feiner. *Computer graphics: principles and practice*. Pearson Education, 2013.

[7] M. Kuderer, C. Sprunk, H. Kretzschmar, and W. Burgard. Online generation of homotopically distinct navigation paths. In *Robotics and Automation (ICRA), 2014 IEEE International Conference on*, pages 6462–6467. IEEE, 2014.

[8] B. Lau, K. Arras, and W. Burgard. Multi-model hypothesis group tracking and group size estimation. *International Journal of Social Robotics*, 2(1):19–30, 2010.

[9] M. Montemerlo, S. Thrun, D. Koller, B. Wegbreit, et al. Fastslam: A factored solution to the simultaneous localization and mapping problem. In *AAAI/IAAI*, pages 593–598, 2002.

[10] M. Quigley, K. Conley, B. 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*, volume 3, page 5, 2009.

[11] R. B. Rusu and S. Cousins. 3D is here: Point Cloud Library (PCL). In *IEEE International Conference on Robotics and Automation (ICRA)*, Shanghai, China, May 9-13 2011.

[12] A. Wallar. Fast occupancy grid generation. `https://github.com/wallarelvo/fogg`. Accessed: 2015-12-12.