Summer 2015

Generalized Mapping and Object Removal

Duncan Campbell
University of Akron Main Campus, dac80@zips.uakron.edu

Generalized Mapping and Object Removal

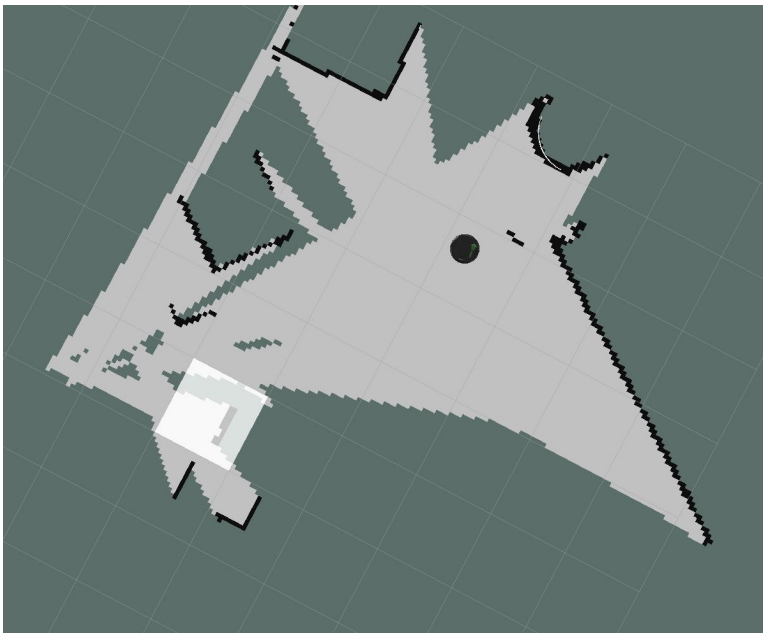
Duncan A. Campbell
Honors Thesis Summer 2015

Abstract

Simultaneous localization and mapping (SLAM) is a problem that has been explored for the past few decades. SLAM deals with the concept of a robot being introduced into an environment in which it has no prior knowledge. Then, through the use of sensors, the robot is able to map its environment while simultaneously determining its position within the given area. While there has been extensive research into the development of methods by which this problem can be solved, not much has been done on what to do with the resulting maps once they are produced. The research conducted deals with maps that are generated of indoor environments where some object such as tables and chairs can possibly change location within their environment, making storing their location unnecessary. There were several methods explored regarding the ability to remove such objects from the environment without unintentionally removing objects that are needed to be kept. The methods and their implementations are then integrated within the Robotics Operating System (ROS).

Introduction

Robotics is an ever changing field that is finding more and more applications in everyday life. They are capable of going into areas where humans are not capable of going and mapping the environment as they explore it. One area that robots have become exceptionally good at mapping and navigating in is inside of buildings. Robots are often capable of going where people are not able to go in the building normally such as under tables and chairs. When the robots are navigating this environment, often they are building up a map of it as they go along. The approach on how the robot will determine its location in the environment and map it is crucial. Mapping yields a lot of information that the robot would have to store for future use. The data would need to be retained in some manner by determining which information is the most critical for long term use.



In order for a robot to be able to generate a map of its surrounding, it must first figure out where it is relative to its surrounding while mapping. This problem is referred to as Simultaneous Localization and Mapping (SLAM). One way this can be approached is through a process called gmapping.

Objects provide obstacles within the maps generated. Hence**,** a method must be developed that is capable of removing the desired objects from the map while still retaining what was considered

critical information about the map. The first portion of this is to determine what was the critical information that had to be preserved when removing items from the map. Since the robot is navigating in an indoor environment, it was determined that the walls are the most critical piece of information to retain and hence the method that was developed focuses on eliminating objects from the data that are not walls within the room.

Background

Simultaneous Localization and Mapping (SLAM) is the basic concept that a robot is capable of both mapping an unknown environment while also determining its location within that environment at the same time. The robot's location within its environment is known as its pose which is composed of the robot's coordinates and orientation. For simplicity, the robot always automatically assumes that where it starts is the origin of the map to ease the process with being in an unknown environment.

There are several different approaches to SLAM, however, the process can be generalized to composing of a few steps. The first step is for the robot to update its position with odometry data that it has collected. The robot then observes the surrounding environment and relates that back to odometry data that it has collected. From the comparison of this data, the robot can determine a probabilistic location of where it is in the environment and can then update the map with the information that it has observed about its outside environment as well. Figure 1 illustrates this process at a high level.

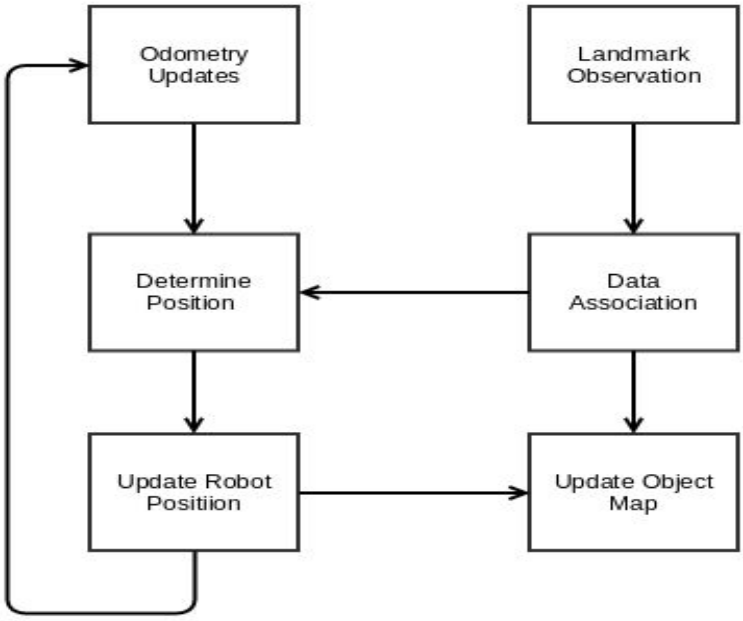


Figure 1: SLAM Process at a High-Level

Gmapping is the SLAM method that is chosen to obtain the map of the environment in which the robot is in. This method falls into the category of particle based SLAM methods and makes use of the Rao-Blackwellized particle filter. There are several steps to this algorithm. The first is that it starts off by generating a distribution of particles each of which have their own version of the layout of the environment. Then the goal is to compare the particles measurements with the actual measurements and assign each particle a weight corresponding to how well the particle estimates the environment around it. This process continues with each new observation, however gmapping optimizes for each successive generation by selectively choosing and generating the particles to choose from the do the sampling.

Identifying the objects that can be removed is the next critical part of solving the problem. For this we choose to take a computational approach using area as the heuristic for deciding whether or not the object should be retained in the map or not. In order to group the points into objects we used an approach called an alpha hull. This algorithm groups points together by their proximity to one another. In each group it is guaranteed that each point will be at least a certain distance or less from one other point in the same group. In order to do this it loops through all the points and if a point is within a certain distance of another it places an edge between them. Then we cycle through the whole graph produce and add each unique point to the set to produce the set of points that then represents an object.

Now for the reason for removing objects from the map. Point cloud data can end up taking a lot of room once the data sets become large enough. Removing objects that are small or otherwise could possibly move allow the data to be stored in a much smaller area. Hence, this acts as a form of compression for the map data that can be done after the robot has gathered the initial map. Another reason for removing objects is so that the map can be a more generalized representation of the environment that the robot is in. This is why we choose to remove smaller objects, because when looking at a map to determine which objects could move or not stay in the same place within the environment, oftentimes it is the objects that have a smaller area from the perspective of the robot. These are objects such as chair legs, table, legs, columns, and other small objects that have a small area when looking at a cross section of the object.

Approach

Our approach to finding objects to remove was to decide what type of objects that we would like to have removed from the map. The first area that we had to address was to figure out the area that we would be mapping. We decided upon mapping the inside of a building as many of the current SLAM solutions are able to produce accurate maps of indoor environments for the solution to work on after they have been produced. Now when looking that the map we had to decide what type of objects should be removed from the map. We decided upon objects that could easily be moved and that have a cross-sectional area should be removed from the map. The reason for this was that if an object could be moved and was small enough then in the future the object was more likely to be moved when a robot went back to go and observe the environment again. Thus we are removing this objects to construct a more generalized map of the environment that can be used by the robot or others in the future.

Now that we have decided what objects that we wanted to remove from the map, we have to develop a way to identify the objects that are present in the room in the first place. The approach that we took to solve this problem was through the use of an alpha hull to identify which points belonged to which objects. For this portion of the problem, we calculated the distance from one point to each of the other points that are located on the map. Then if the distance was less than the width of the robot we added an edge on a graph connecting the two points together. The next part was to take the graph that was generated and the group the points into sets representing which object that they belong to. The final step on this stage is to take the sets of points and perform a convex hull on each set of points to derive the polygon that represents the maximum possible area that the points could cover.

The final stage to this process of course is when we remove the objects from the map. In this stage we take each of the generated polygons and compute the area that each of them occupies. We then compare that to the threshold area we have set for objects that we want to hold onto. If the object's area is less than the one that we specify to be the threshold we then remove the object from the map.

Implementation

For the implementation of this solution we utilized several different technologies to aid in its development. For the overall project a software layer known as the Robotics Operating System(ROS) was used and installed on an installation of Ubuntu 14,04, ROS provides several features which are useful in many robotics applications. The first of these features is a method for interprocess communication through the use of a publisher/subscriber model. This allows each process or node in ROS terms, to communicate with one another and pass data around. Another feature that is provided by ROS is a common build system for building packages to interface with ROS. This allows for all ROS packages to be made from a common framework and allows others to share the packages that they have created with one another. The third benefit to using ROS is that there are many packages that are already available ready to be used to interface with your robotics application.

Another software package that was utilized throughout the creation of this project was the Gazebo robotics simulation software that is provided by default with ROS. This software allows us to simulate many aspects of our robot without having to go to a real robot which is a costly investment. Through this simulator we were able to simulate the data that would be produced by sensors we would be suing when mapping such as odometry data from encoders and accelerometers, as well as sensor data from things such as LIDAR and cameras. Gazebo also provides a visualization of the environment that the robot is in to go along with the output that is produced by the simulation. Figure 2 shows the visualization that ROS provides during simulation. In this figure an office layout is shown that the robot could possibly navigate in order to generate a map.

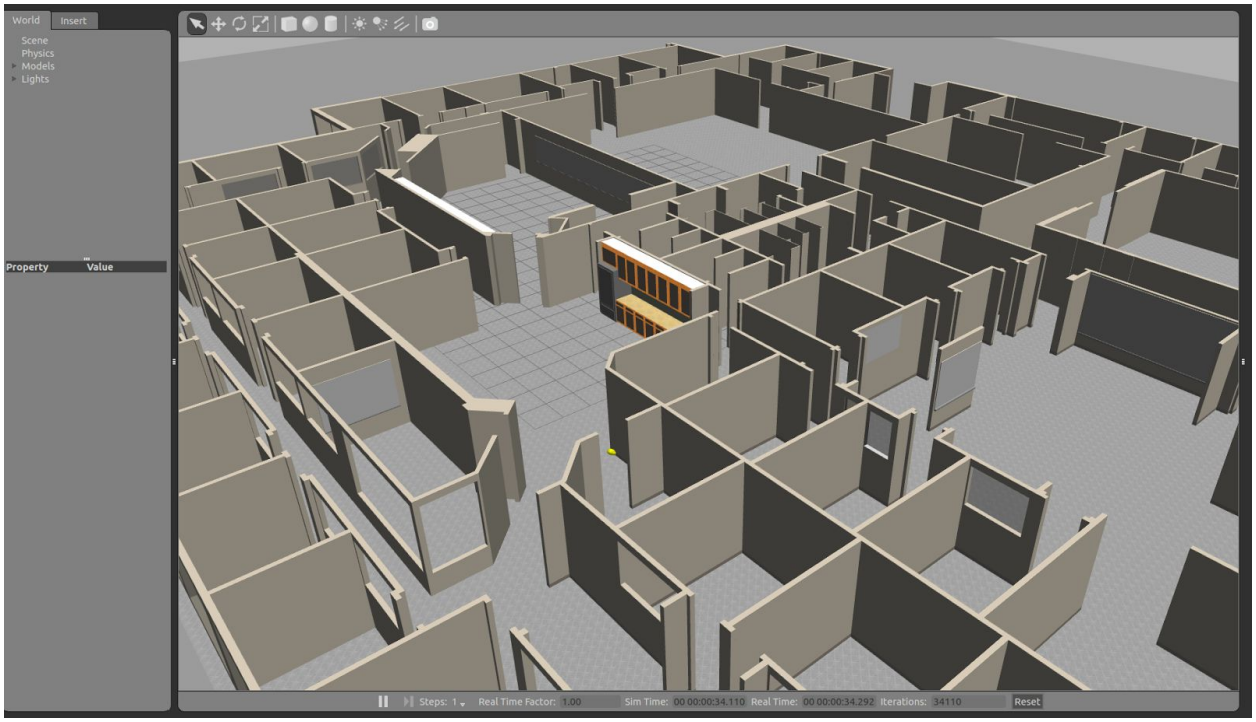


Figure 2: Screenshot of Gazebo editor and a possible world the robot could exist in

We choose to implement the algorithm as a part of a ROS package that uses a node to retrieve the map data as an occupancy grid and output a generalized point cloud map for the objects. The alpha hull algorithm combined with the method discussed in the approach method are implemented in python as a part of this node to process that data. The node utilizes the rospy package to receive and send of the messages and the maps. Figure 3 shows a small python excerpt that handles that processing of the information once it has been converted into points.

```
def findClosestPoints(point, pointList):
    points = list()

    for i in range(0, len(pointList)):
        if getDistance(point, pointList[i]) < 10:
            points.append(i)

    return points

def unionClosestPoints(setNum, setPoints, pointMap):
    for i in range(0, len(setPoints)):
        if pointMap[setPoints[i]] == -1:
            pointMap[setPoints[i]] = setNum
        else:
            oldSet = pointMap[setPoints[i]]

            for d in range(0, len(pointMap)):
                if pointMap[d] == oldSet:
                    pointMap[d] = setNum

    return pointMap
```

```
def getSets(pointList):
    pointMap = [-1 for x in range(len(pointList))]
    nextSet = 1

    for i in range(0, len(pointList)):
        setPoints = findClosestPoints(pointList[i], pointList)
        pointMap = unionClosestPoints(nextSet, setPoints, pointMap)
        nextSet += 1

    numSets = len(set(pointMap))
    return pointMap, numSets
```

Figure 3: Python code used to process the point data in the map processing node

The end result of combining all of these different components and technologies results in a system that is capable of mapping its environment and, to a limited extent, producing a generalized map of the area that it is in. This system was designed to be run on a Turtlbot, the model for which is provided as a part of ROS. All of the data manipulation for the process can be conducted onboard the robot. The general flow of the system is the SLAM component reads information from the sensors and produces a map. This map is then passed to the map processing node which then produces a generalized map. The system as a whole is displayed in Figure 4 which shows the overall layout of the system.

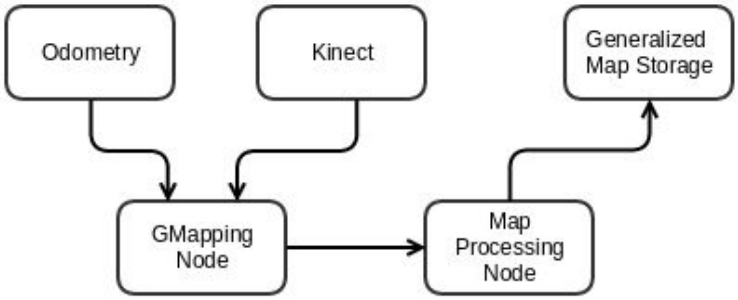


Figure 4: Overall systems diagram for the robot

Results

The overall algorithm was successful at isolating objects and removing the ones that were considered small enough, however there were limitations on the objects that it was able to isolate due to the method being used. Since points have to be a certain distance from one another to be added to a group these in effect creates a form of resolution as to how fine the detail is of the map. For the threshold distance the width of the robot was chosen because the robot needs to be able to navigate around the object in order to detect it. So if the robot was not able to navigate in between two objects they would be considered one object under this algorithm. Now in order to illustrate the results in a more presentable fashion we will refer to Figure 5 as a reference to explain the results and the limitations exposed by these results.

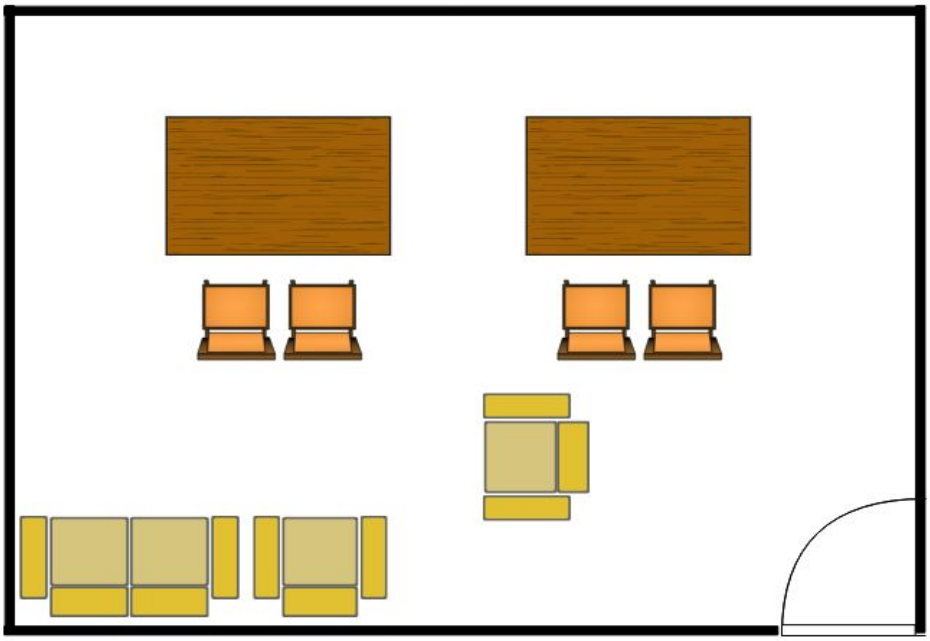


Figure 5: Possible room that could be scanned by the robot

When the robot scans the room above it locates 26 different objects that it thinks are in the room. Now when looking at the room most people would say there are 9 but the robot detects 26 because from its point of view each table leg and chair leg are one object, and the room as well is one object. Now the reason it detects 26 and not 28 is because it assumes that the couch and arm chair are a part of the wall since it can not navigate in between them. This situation illustrates the main limitation of this algorithm which is that it is not always able to separate out all of the objects and may merge certain ones together producing a larger object.

When this was run through the map processing algorithm the tables and chairs were removed since the area that each of their legs covers is smaller than the threshold area that was set to be removed. The armchair was kept as the robot assumed that it was large enough that it would not be moved around later on. The robot also did not remove the walls, couch, and armchair were not removed because they are considered one large object together in this room. This room was one of several tested although it was chosen since illustrates the limitations of the system most accurately.

Future Work

There are several areas that could be further improved upon in the methods that are explored by this paper. Much work is still yet to be done in the production of systems that are capable of producing generalized maps. This paper shows just one attempt at producing such a method and many more have been done or have yet to be done.

In regards to the method used in this paper, its implementation could be optimized to be more efficient overall. Right now it is currently written in python which is an interpreted language. It

could be written in a language like C or C++ and interfaced with ROS to yield a more efficient system. Another area that could be expanded upon in this system is to repurpose the algorithm to detect the outer part of the room or the area that is located in. This would in essence create a general floorplan of the room that the robot has mapped and would represent the map at its most generalized form. Other methods that could be explored but were not for this paper were characterizing the shapes of objects and using those to classify which object could be removed or not.

Conclusion

In conclusion, the method that was explored in this paper was effective for developing a generalized map and reducing the amount of memory that was required in order to store the map. There are other methods that could be explored for developing a generalized map, but choosing to focus on area as a the heuristic showed that deciding on just area alone is not enough for removing all object that could be moved from the map. In the development of this method, it was also revealed that a modified version of the approach could be used to generate the outline of the room which would yield the most generalized form of the map. There is still much work that can be done to improve upon the methods that were explored in this paper in the future and there is great prospect for practical applications of this methods such as floor planning, and maps the robot can use for later exploration.

References

1. Durrant-Whyte, Hugh, and Tim Bailey. "Simultaneous localization and mapping: part I." *Robotics & Automation Magazine, IEEE* 13.2 (2006): 99-110.
2. Bailey, Tim, and Hugh Durrant-Whyte. "Simultaneous localization and mapping (SLAM): Part II." *IEEE Robotics & Automation Magazine* 13.3 (2006): 108-117.
3. Akkiraju, N et al. "Alpha shapes: definition and software." *Proceedings of the 1st International Computational Geometry Software Workshop* Sep. 1995: 63-66.
4. Thrun, Sebastian, Wolfram Burgard, and Dieter Fox. *Probabilistic robotics*. MIT press, 2005.
5. Dudek, Gregory, and Michael Jenkin. *Computational principles of mobile robotics*. Cambridge university press, 2010.
6. Riisgaard, Søren, and Morten Rufus Blas. "SLAM for Dummies." *A Tutorial Approach to Simultaneous Localization and Mapping* 22.1-127 (2003): 126.
7. Bajracharya, Suraj. "BreezySLAM: A Simple, efficient, cross-platform Python package for Simultaneous Localization and Mapping (thesis)." (2014).