Autonomous Simultaneous Localization And Mapping

Michail Shaposhnikov, Mustafa Abban, Jake Crabtree
CS309

May 12, 2017

1 Abstract

A discussion of the methodology by which a robot may autonomously explore and map a previously unmapped area is presented in the context of the tools provided by the Robot Operating System project and the BWI Segbot project of the University of Texas at Austin Autonomous Intelligent Robotics laboratory. The approach we took was based on the existing gmapping library which constructs a map from a stream of odometry data as well as our own nodes that were responsible for moving the robot around in an unmapped environment. Our idea was to have the robot use a greedy algorithm to move forwards if it can, then rotate left or right by the minimal amount needed for it to pass through without colliding into anything. We attempted to make use of the actionlib, costmap2d, hector mapping, and point cloud libraries. Experimental results showed that navigating in an unmapped area presents a chicken-and-egg scenario in which cyclical dependencies prevent a straightforward or standard approach to navigation.

2 Introduction

2.1 What is Autonomous SLAM (Simultaneous Localization and Mapping)?

The current process for mapping a new area in the BWI is manually maneuvering the robot through the entire environment and visually confirming that the entire area is mapped. While this procedure has accurately mapped many rooms and floors at the University of Texas campus, there are several drawbacks to this methodology: notably, how it is very time-consuming for researchers to do this for each and every new environment the robot is exposed to, and also, how it makes the robots reliable on humans to provide its navigation, losing their autonomy. What SLAM does is it allows for the placing of an autonomous

robot in an unknown area and then navigate the area, mapping it and maintaining its own localization within this newly mapped environment. Since this area is completely unknown when the robot begins to navigate, no prior knowledge of the environment or assitance from humans is necessary, remedying the main issues with the aforementioned solution. [3] One of the most popular algorithms for solving the SLAM problem is what are known as Rao-Blackwellized particle filters (what is used in the Gmapping package used in both the BWIBots and this project, the results of which can be seen below). The approach gives each particle an individual map of the environment, which are narrowed down by both considering the change in position and in observation (via its sensory inputs) of its surroundings of the robot within the environment. In addition, the process only selectively resamples and eliminates particles in order to reduce the chance of running out of possible particles and causing the robot to become completely lost. A study on these methodologies conducted by Professor Giorgio Grisetti of the University of Freidburg with their own robots traversing through the university's campus, MIT, and Intel Labs indicate that this methodology is much more effective than its predecessors, and for this reason, was selected for use in this project. [4]

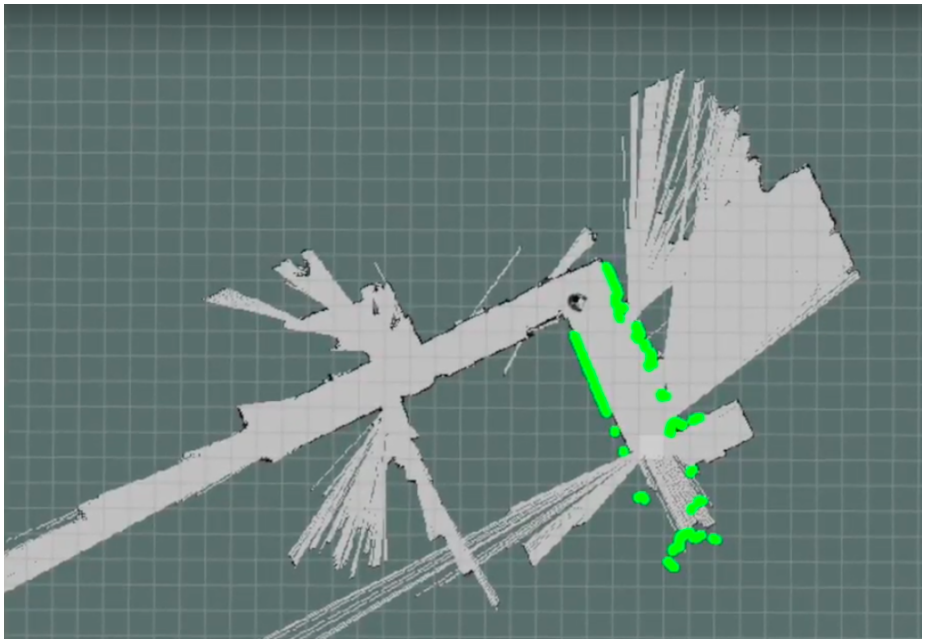

Figure 1: An example map produced by the gmapping package.

3 Background and Related Work

3.1 Behavior-Based Exploration

What we wanted to accomplish was the ability for the robot to use only local odometry and laser scan data to make reach the global goal of exploring the entire space of open area available to the robot. Now, we did not care in which order the robot explored each portion of the open area. We simply wanted it to exhibit the *behavior* of exploring each section of the open area. This type of approach where we are not concerned with the individual actions that the robot takes but rather the overall perceived behavior is called behavior-based exploration. The advantage of behavior-based exploration is that it only depends on local information (such as the location of nearby obstacles) to make a decision. In contrast, the current in-use planning-based navigation of the BWI segbots relies on a global costmap and a local costmap to decide the best series of navigation goals that, in sum, would lead to a successful plan.

3.2 Bug Algorithms

One specific example of a class of behavior-based exploration algorithms that we perceived to be particularly useful to our endeavor are known as "bug algorithms". These are completely based on data obtained from the sensory inputs of the robot's immediate surroundings (e.g the Kinect sensors, cameras, etc.), being very useful for an autonomous SLAM implementation that inherently begins with no maps. Bug algorithms all consist of determining, via these local inputs, where walls of impassable terrain occur and then moving by following along with these walls to the issued navigation goal. Some strategies for reaching the navigation goal were discussed in a lecture by Professor Choset of Carnegie Mellon University, one of which, the simplest approach, follows along the wall to completion and then returns to the path for its original navigation. Another, more robust approach creates a line between the agent's inital position and its navigation goal, follows the path created by that line until it encounters a wall, traverses the wall until it re-enters the path of the line, and repeats until the navigation goal is reached and another can be reached. [2]

Most modern robotics systems, however, the robot possesses sensors that allow detection of range for objects in the distance, allowing for one of the most popular of these algorithms: the tangent bug algorithm. This can be broken up into a two parts, the first of which is: computing the ranges of all segments in view, estimating and taking the path that minimizes the distance to the end goal(similar to the second bug mentioned earlier). This process is repeated until the goal is reached or the path taken increases the distance from the goal point, in which case, the boundary-following behavior is invoked. In this phase, the robot follows the boundary until the goal is reached, the entire blocking object is circumnavigated (and therefore the goal is unreachable), or when the shortest distance between the boundary the robot is following and the goal exceeds the

shortest distance between the obstacle's center and the navigation goal, and, in this case, returns to the first state.[2]

4 Technical Approach

4.1 Greedy Exploration

We believed the most practical solution was to approach the problem in terms of a greedy heuristic. The segbot should have a bias towards going forward because that is the most optimal case when trying to cover as much new ground as possible. However, it is important to note that effective exploration can not be achieved without a sufficient understanding of the segbot's current environment. This inspired us to break the problem down into two. First we must be able to keep track of what is in front relative to the segbot. This is a notable factor of our approach because forward is the only movement we make besides rotating. However, a problem occurs after the segbot has changed orientation. For example, a segbot facing east and one facing north both develop the same cost map. So one must know where to check on the cost map relative to the segbot's current orientation. This can be done with the following formulas:

$$y' = y + 2r(cos(\theta))$$

$$x' = x + 2r(sin(\theta))$$

The results from these are the coordinates on the cost map for the new area in front of the segbot. Once this has been set, we can go onto interpret our environment in order to create decisions. The process is done recursively as follows:

Data: x, y, r, θ Where x - x-coordinate of forward area on cost map, y - y-coordinate of forward are on cost map, r - radius of robot footprint, θ - robot's pose rotation in degrees from when it started, 0 being the initial starting state
begin
  y' = y + 2*r*cos(θ); x' = x + 2*r*sin(θ); bool occupied = areaOccupied(x',y', $1.5 * \pi * r^2$); **if** *occupied* **then**
    if $x >$ *map.size/2* **then**
      return explore(x,y,r,θ - 10);
    end
    else
      return explore(x,y,r,θ + 10);
    end
  end
  else
    return Go forward;
  end
end

Algorithm 1: Exploration Decision

The result of the algorithm is visualized in Figure 2. Essentially what occurs is the segbot will keeping rotating away from occupied spaces until reaching a safe goal. Alongside this, the segbot is also updating any blind spots that had developed as it rotates.

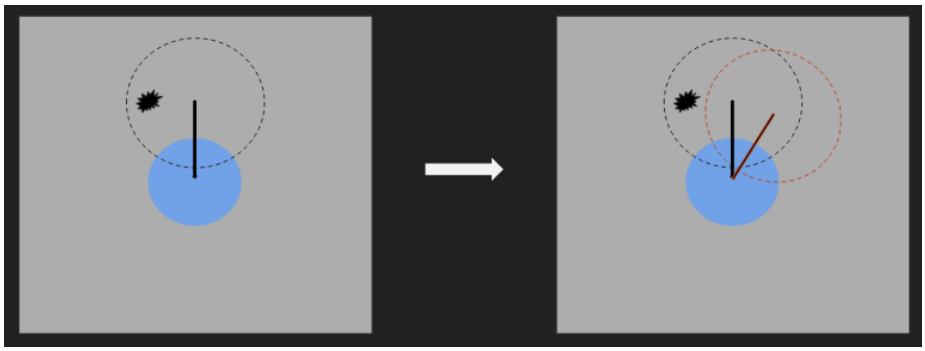

Figure 2: Determining orientation and defining a new forward position.

5 Experiments and Evaluation

5.1 Challenges

Moving the segbot with the vanilla ROS navigation stack without a pre-existing map of the robot's environment posed many challenges for the implementation of our algorithm.

We wanted the segbot to map out an area and in order for the *gmapping* package to construct a map, the robot needs to physically move to new locations in order to aggregate new laser scan data. The simplest solution would be to use a *SimpleActionClient* to issue a *MoveBaseAction* to a given location. However, the default navigation stack relies on a global map in order to test whether the given navigation goal would result in a dangerous maneuver. When we run *gmapping*, we don't have access to this map because we are building it.

The next logical step would be to simply use local costmap data to determine where out of the nearby locations pose threats to the robot and move to the empty locations then repeat that process over and over. However, without the ability to issue navigational goals using *SimpleActionClient*, the only other way we knew how was to publish velocity commands directly to the */cmd_vel* ROS topic. However, in doing so, we circumvent the built-in obstacle avoidance countermeasures and safety checks that ensure the robot does not endanger itself or any person that happens to be around it.

We will discuss how we propose to solve this chicken-and-egg problem between being able to move the robot and being confident that our move is safe in the *Conclusion and Future Work* section.

5.2 A Modified Solution

In order to comply with and be able to use the BWI's pre-existing navigation stack, a much simpler algorithm was developed that emulates the spinning behavior of the popular consumer cleaning product, the Roomba, which is as follows:

```
begin
    for i = 1 to 10 do
        Explore 10 times before stopping
        attemptToMoveOneMeterForward(); while can't successfully
        moved forward do
            turn()Turn to try new direction to avoid obstacle
        end
        if once every 5 moves then
            spinAround(); Spin around 360 degrees to gather more
              accurate readings for gmapping
        end
    end
end
```

Algorithm 2: Theoretical Exploration Navigation

5.3 Results

After running this algorithm through the 3rd Floor Robotics lab of the GDC several times, the effect of the navigation goal's size (initally 1 meter) immediately became apparent. With higher distances, the robot was able to traverse significantly further distances, but became delocalized significantly quicker, whereas with a smaller navigation goal (around 0.5 meters), the robot traversed significantly less horizontal distance, but remained localized with valid goals for much longer. This effect is due to the purely random nature of yaw selection in this algorithm, leading to opposing yaws that significantly decrease the net distance traversed that is significantly enchanced in the case where more yaws are selected prior to the goal being unrecognized by the map structure and the fact that with a larger navigation goal, it is more likely to be placed outside the bounds of the previously mapped environment. From this, it can be concluded that in a truly autonomous SLAM algorithm, goals and attempts to traverse the environment should not be randomly selected, but based on where the algorithm has already been and has yet to be, as the greedy algorithm described in *Section 5.1* implements.

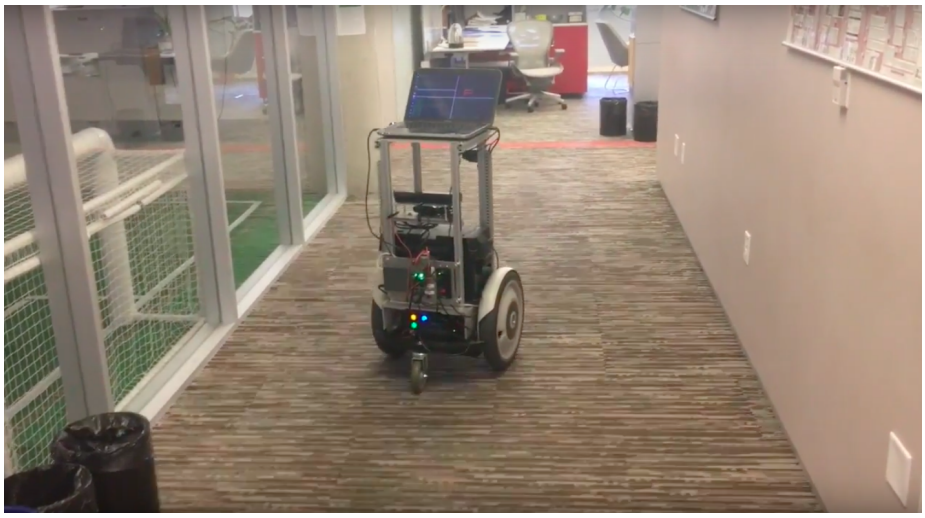

Figure 3: Segbot exploring local region.

6 Conclusion and Future Work

6.1 Refining the Problem

We have learned that in order for a robot to autonomously explore its own environment (autonomous simultaneous localization and mapping) it needs to solve a chicken-and-egg problem. It needs to feed data from its odometry sensors into a package that analyzes the point cloud of nearby obstacles and converts that data into a local cost map with probability distributions of potential obstructions.

As for sending navigational goals without a global map, it appears that the ROS navigation stack offers a plugin system that allows ROS nodes that follow a certain API to be able to be drop-in replaced or even combined with existing ROS navigation nodes. This method would allow the majority of the navigation stack to remain the same while still modifying the decision tree to allow the robot to map areas of interest. From preliminary reading, it appears that this process includes a substantial amount of editing configuration files and various registration files across the ROS stack. Nonetheless, this solution appears to be the most reasonable because it does not reinvent the wheel and it requires the least amount of code.

6.2 Future Work

Advanced Obstacle Avoidance There remains a lot to be done in this domain before a truly autonomous and intelligent agent is able to map new

areas. For instance, there are still no robust obstacle avoidance mechanisms on the BWI Segbots. Currently, there are only 2D lidar and Kinect sensors. There are no 3D implementations or anything that would protect the robot from falling down stairs for instance. Therefore, before this project can continue, we need to give the robot a greater perception of its surroundings. This can be accomplished both with additional hardware (depth sensors, pan-tilt cameras, etc.) or with additions to the software (3D cost maps, etc).

Correctness and Completeness Next, while the robots are able to explore the area in front of them, a human must still act as the fitness function in order to check whether or not the robot correctly identified all the areas that are needed for operation. This is because the robot's sensors may not correctly identify certain hard to reach areas or areas that require advanced interaction to access such as doors or elevators. This is a fault tolerance problem that can be approached by thinking of the area to map as a maze. Now, we can apply maze-solving algorithms that would simply infinitely look for an exit until they map the entire explorable region. Lastly, when the maze solver runs out of options to explore and terminates, we know we have mapped the entire area. What remains to create is a framework for checking the correctness of these solutions. This can be accomplished by testing a variety of maze solving algorithms and perhaps deploying several of them at once and creating some sort of weighted average of the different maps created by the various algorithms.

Labeling and Semantics Of course, without some semantic meaning, a map of an area by itself offers little benefit to higher level planning. Consequently, we have to develop a methodology for easy and streamlined labeling and captioning of a given map. Our current vision is an interactive web-based application that we can share with people who are familiar with the layout of a given region. They would add labels to the image (similar to tagging faces on Facebook). Then, the labeled image would be analyzed and a .yaml file with the proper name-location parings would be generated from these labels in order to allow the robot to process high-level navigation requests between various locations in the environment.

6.3 Future Applications

The application of autonomous exploration is crucial for the deployment of the BWI segbots to new and varying locations. This technology would allow researchers to test their own experiments in more diverse locations in order to prove that their projects are not restricted to the GDC lab spaces. In fact, once the robots have generated a map of their environment, virtually anywhere can be mapped: allowing for, although perhaps not with segway bots, deep-sea and space exploration, as well as opportunities for autonomous vehicles, air, and spacecraft that cannot only traverse the previously inaccessible environments, but map them for all of humanity to enjoy and learn from.

References

[1] A solution to the simultaneous localization and map building (SLAM) problem (http://ieeexplore.ieee.org/abstract/document/938381/)

[2] Robotic Motion Planning with Bug Algorithms (http://www.cs.cmu.edu/ motionplanning/lecture/lec3.pdf)

[3] A Solution to the Simultaneous Localization and Map Building Problem (https://roboticsclub.org/redmine/projects/colony/repository/revisions /1613/raw/branches/scout/SLAM/IEEEXplore.pdf)

[4] Improved Techniques for Grid Mapping with Rao-Blackwellized Particle Filters (http://www2.informatik.uni-freiburg.de/ stachnis/pdf/-grisetti07tro.pdf)

Code Cited

- **BWI** https://github.com/utexas-bwi/bwi
- **Segbot Navigation** https://github.com/utexas-bwi/segbot/tree/master/segbot_navigation/launch
- **Hector SLAM** https://github.com/tu-darmstadt-ros-pkg/hector_slam

GitHub Repo

- https://github.com/MishaShapo/bwi_a_slam