Final Project Report

December 3, 2018

Daniel Boreman
Lucas Frey
Mazen Alotaibi

Abstract

...

Contents

1 Discussion of the exploration problem

In this problem we are designing an exploration package utilizing the gmapping and nav_bundle packages to allow a simulated robot to explore an unknown environment. Our algorithm will have to set waypoints to move our robot towards the unexplored areas while avoiding obstacles. The robot should be reasonably robust to noisy odometry and mapping data, and it should be able to recognize when waypoints cannot be reached.

2 Discussion of your gmapping and nav_bundle package implementations

From the gmapping bundle we are only using the occupancy grid. This grid is used to find ”frontier” points (points between explored and unexplored areas). We are also reading the map meta data which gives us the resolution of the occupancy grid in meters/pixel. This gives us the ability to transform the occupancy grid data into Cartesian coordinates. The map is also saved when a waypoint is generated and used to verify that the robot is staying in known areas only, ensuring that the robot doesn't run into walls even if it didn't know about them before it calculated it's path.

From the nav_bundle package, we are using the waypoint commands: twist, base_link_goal, path_reset, move_base_cancel, and ready_pub. Clear and cancel are used to have the robot only pursue a single waypoint at a time. Waypoints generated using the occupancy grid which are then translated and rotated into the robot's local coordinate system then set as a waypoint using Twist.

3 Discussion of your waypoint allocation algorithm

Waypoints are generated procedurally using an 61x61 filter that scans the frontier points on the occupancy grid. The robot's exploration policy defines frontier points as being known unoccupied locations where the robot would end near unknown locations. This filter only selects points that are centered on an explored point, have no obstacles within a specified distance from the center, have a threshold percentage of unexplored cells, and aren't where the robot has been before. These cells are then weighted by the percentage of unknown cells and euclidean distance from the robot. It then uses an A* algorithm to determine if there is a known path from the current location to the candidate location. Validating the path allows the robot to exclude candidate points that would be outside of the map or within obstacles.

If the robot enters a region that was unexplored when the waypoint was *created*, it will clear the waypoint queue, cancel the current waypoint, turn 360 degrees, back up 1.5m, and generate a new waypoint. With the current implementation of the nav_bundle, the robot can select paths that pass through unknown locations. If the robot then passes though the unknown location and discovers that there is a wall blocking the path, the robot will not reroute in order to find the proper path. Instead, the robot will simply crash into the wall. To prevent incorrect path planning through unknown locations, we save how the map looked when the nav_bundle chose that path, and tell the robot to stop and reroute if we reach a location that was previously unknown, making our robot's path robust to new information.

4 Analysis of your algorithms exploration performance

The algorithm has managed full coverage in all provided maps. This algorithm is far from perfect, periodically getting stuck (although it is able to correct itself), and periodically choosing waypoints outside of the map. The algorithm can struggle to come up with waypoints in a reasonable amount of time (typically 20 seconds) due to the need to run A* after a potential candidate is selected.

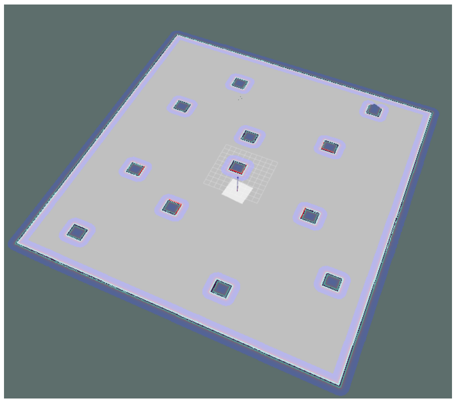
Algorithm 1 FindNextWaypoint

```
1: procedure FindNextWaypoint
2:     Input
3:         $map$:                      The current occupancy grid
4:         $window\_size$:             Width of the sliding window used to generate candidate points
5:         $stride\_length$:           Row much to shift the sliding window with each iteration
6:         $visited\_locations$:       Grid with same size as map representing the locations we have been to
7:         $robot\_loc$:               Currently location of the robot
8:         $avoidance\_radius$:        Radius around candidate waypoint that should not contain walls
9:         $empty\_radius$:            Radius around candidate waypoint that should be empty
10:        $RESOLUTION$:               How many meters per cell
11:    Output
12:        $candidate$                 The best potential waypoint to go to next
13:    $potential\_candidates \leftarrow$ []
14:    for every $center\_cell$ in $map$ that is greater than $stride\_length$ apart from each other do
15:        $skip \leftarrow false$
16:        for every $neighbor\_cell$ in a $avoidance\_radius$ away from $center\_cell$ do
17:            if alreadyVisited($visited\_location$, $neighbor\_cell$) then
18:                $skip \leftarrow true$
19:                break
20:        if $skip == true$ then
21:            continue
22:        for every $neighbor\_cell$ in a $empty\_radius$ away from $center\_cell$ do
23:            if getMapValue($visited\_location$) $\neq 0$ then
24:                $skip \leftarrow true$
25:                break
26:        if $skip == true$ then
27:            continue
28:        $cell\_sum \leftarrow$ sum(all cells within $window\_size/2$ from $center\_cell$)
29:        $potential\_candidates$.append($center\_cell$)
30:    $candidate \leftarrow$ sorted($potential\_candidates$)
31:    $candidate \leftarrow potential\_candidates$.pop()
32:    while not reachableByAStar($robot\_loc$, $candidate$) do
33:        $candidate \leftarrow potential\_candidates$.pop()
34:        $best\_loc = candidate$
35:    return convert_row_col_to_coord($best\_loc$, $RESOLUTION$, $map$)
```

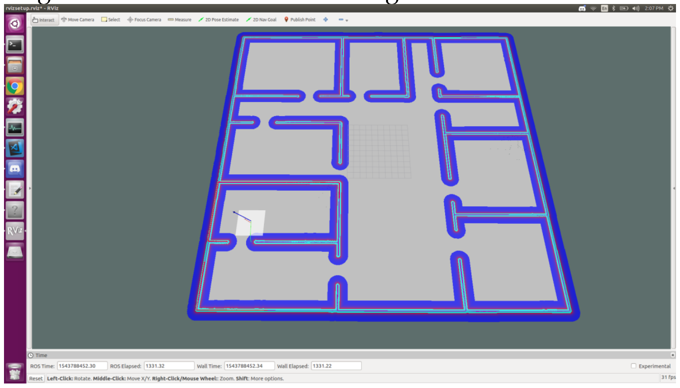
The algorithm tended to get the robot stuck in one area, ensuring that every cell in the immediate area. This behavior causes the exploration process to take a long time in the maze map if it selects a point on the other side of a wall potentially causing long travel times.

4.1 Did it result in full coverage of the environment (provide a screenshot from rviz)?

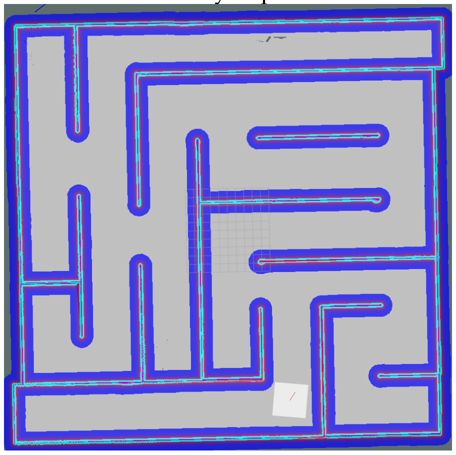
This algorithm resulted in full exploration of the robot's environment with a reasonable success rate.



The many dots map was the first map that the robot completed, it completed the map in around 20 minutes. Euclidean distance in this map is an accurate way to approximate path distance since the map isn't divided into rooms, making it an ideal scenario for this algorithm.



The office map saw completion times of around 22 minutes. This was largely due to the euclidean distance heuristic that had the robot fully explore each room before moving on.



This algorithm struggled with the maze file since it had a habit of setting waypoint on the other side of a wall making it drive back and forth around the entire map, making little but steady progress.

4.2 Provide suggestions on how your waypoint allocation algorithm could be improved.

The algorithm should use an A* search in order to find the nearest waypoints instead of using euclidean distance. This should make the exploration time faster since it would travel through fully explored areas less often in order to reach new locations. Computationally expensive functions (such as A*) could be rewritten in C++ in order to cut down on computational cost. The robot's policy for getting unstuck could also be improved. Currently we assume that the robot gets stuck, it is facing a wall and can therefor get unstuck by backing up enough. However, it is not always the case that the wall is in front of the robot, and backing up could end up moving the robot into a wall. Instead, a better policy would be to turn and move away from the closest wall whenever the robot is stuck.