Oregon State University
College of Engineering

ROB 456

Final Project

Joel Goodman
Corwin Perren
Dylan Thrush

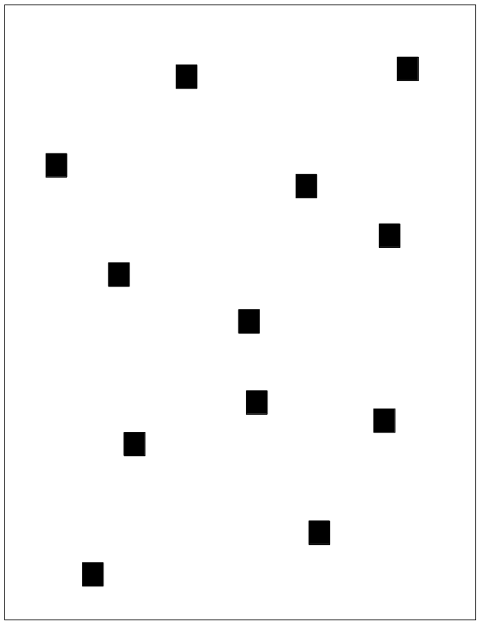
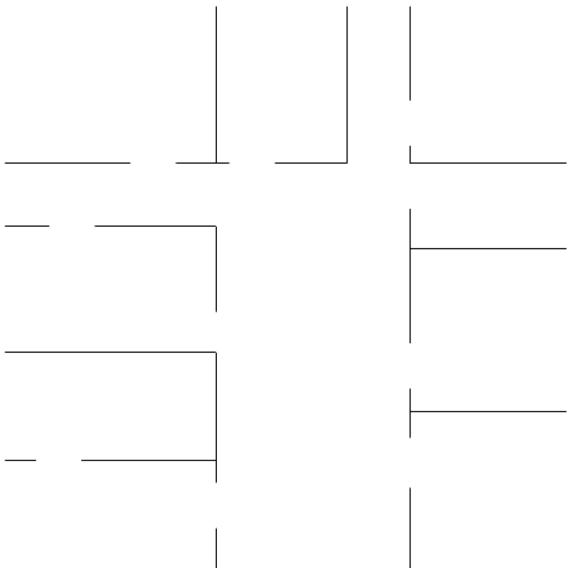


Figure 1: manyDots

1 Discussion of Exploration Problem

This project was an opportunity to showcase the student's understanding of some of the core tools they have been exposed to during the course of the term. The backbone of this being the software tools contained in ROS. Given a robot that has no clue as to where it is or what the map of it's world looks like, it was the team's duty to write an exploration policy for the robot such that no reachable part of the robot's world is left unexplored. There were three worlds provided that the robot was to explore:

- manyDots (pictured in Figure 1), which was the default world and one that the teams has seen before in a previous assignment.
- officeWorld (pictured in Figure 2a), which roughly approximates the layout of a floor in an office building.
- mazeWorld (pictured in Figure 2b), who's title says it all. It is a labyrinth that puts the team's algorithm through its paces.


(a) officeWorld

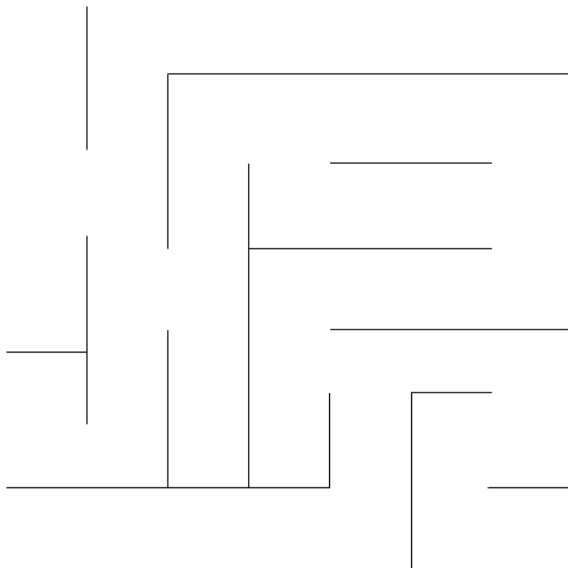

(b) mazeWorld

Figure 2: Worlds ROB 456 Teams Were Given To Explore

2 ROS Implementation

As our team was having many issues getting the provided nav_bundle and gmapping setup tuned properly, we decided to start with examples provided via the navigation_stage package. We also reincorporated some of the setting configurations provided via the ROB skeleton project.

To start, we removed the simple_navigation_goal nodes and started with a slam_gmapping_with_5cm example using move_base that was provided as an example in the navigation_stage package. Then, to get the costmap settings to be useful in a simulated environment again, the yaml configuration files provided via stagebot_2dnav were reassigned to this new nav_bundle launch. To deal with the issues we were encountering in path planning, we opted to use the default planning config file provided with the example package, while changing the map update rate back to five seconds to make navigation possible and rviz useful for debugging. Additionally, we used the default slam config provided by the example as well, which seemed to provide as adequate of a map as the skeleton version.

Outside of these changes, we also increased both the robot footprint and obstacle range in the yaml files to help the robot avoid walls in doorways and corners more easily. The resulting configuration seemed to provide more reliable obstacle avoidance and path planning.

3 Waypoint Allocation Algorithm

The summarized version of the algorithm is that it cuts up the space around the robot into distinct chunks, counts the number of unvisited positions in each chunk, and then chooses the central point of the chunk with the most unvisited points to travel to.

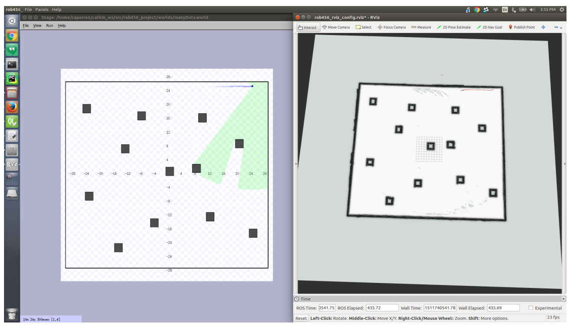
In more detail, the process begins with converting the occupancy grid provided to us by the slam processor into a 2D array using numpy. This makes it easier to access the grid positions by more conventional programmatic methods. We then take a chunk around the robot in the occupancy grid space corresponding to a starting box size. The box size will end up dynamically expanding over time if it is later determined that there are no valid directions to go within the bounds of the current box size. With each box, we slice the occupancy grid values out of the main array corresponding to cardinal direction chunks surrounding the robot. Within each chunk, we sum the number of unvisited points and also store the center point of the chunk in terms of the main occupancy grid's array indices.

To determine which direction to go, all that is needed is to loop through the possible directions and choose the one with the most unvisited points, taking into account a minimum threshold value so that we don't move to a location that is inside pillars, for example. Up to this point, the algorithm would work in ideal conditions, but the simulation space is still far from ideal. To account for this, there are multiple error handling methods to deal with problems that might arise. For one, if a waypoint is set and the navigation system errors out because it is unreachable, the algorithm will allow for two retries of that cardinal direction. If it still fails, the direction is removed from the options to choose from and it defaults to the next direction with the largest count. In the case where these options still fail, a random direction is chosen.

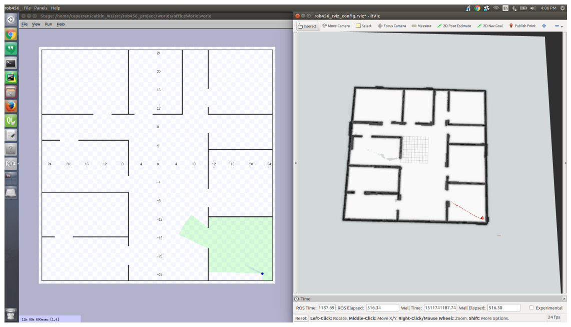
All of these choices benefit us because when the algorithm chooses a point outside of the map walls (which is a common occurrence), and the map is completely closed in already, it will eventually choose a point that isn't outside through random selection. A final fallback is a total system watchdog. This keeps track of the time since the last waypoint callback happened, and then can manually send a movement command to trigger a new callback in the case the robot traps itself so badly that it can't move. For this watchdog, a cmd_vel message is sent instead of a waypoint, which is what allows it to hopefully move out of a corner or stuck location.

4 Evaluation of Team's Algorithm

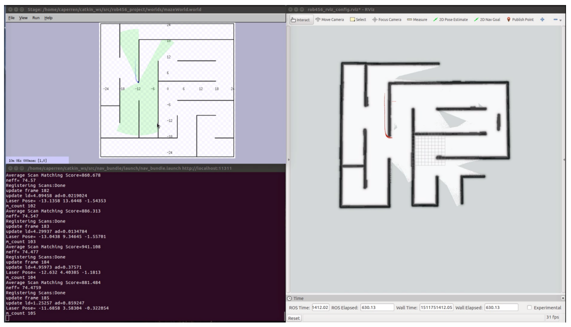
As can be seen above, the team's algorithm navigates the manyDots and officeWorld maps successfully, while consistently completing between a third and half of the mazeWorld map. For the fully completed maps, all areas on the map are explored, which was the primary objective of the project. In addition to successful navigation, the alterations that the team



(a) manyDots Solved



(b) officeWorld Solved



```
/home/caperren/catkin_ws/src/nav_bundle/launch/nav_bundle.launch http://localhost:11311
Average Scan Matching Score=860.678
neff= 74.57
Registering Scans:Done
update frame 182
update ld=4.09458 ad=0.0219024
Laser Pose= -13.1358 13.6448 -1.54353
m_count 102
Average Scan Matching Score=886.313
neff= 74.547
Registering Scans:Done
update frame 183
update ld=4.29937 ad=0.0134784
Laser Pose= -13.0438 9.34645 -1.55701
m_count 103
Average Scan Matching Score=941.108
neff= 74.477
Registering Scans:Done
update frame 184
update ld=4.95973 ad=0.37571
Laser Pose= -12.632 4.40385 -1.1813
m_count 104
Average Scan Matching Score=881.484
neff= 74.4759
Registering Scans:Done
update frame 185
update ld=1.25257 ad=0.859247
Laser Pose= -11.6858 3.58304 -0.322054
m_count 105
```

(c) mazeWorld Partially Solved

made to the ROS node parameters improved the robot's ability to avoid obstacles. However, the implementation is not without its shortcomings, as are easily noticed in its traversal of the mazeWorld map.

Firstly, the most noticeable issue is that the SLAM algorithm that the team was given to utilize has a tendency to leave the boundaries of some walls and obstacles slightly fuzzy. This is doubtlessly a consequence of artificial odometry noise added into the results, and has the effect of making the fully mapped world look just slightly out of focus.

Secondly, the robot is given a policy to follow that can sometimes lead to situations where it revisits the same general area a few times before other more obviously unmapped areas are visited. The reason for this is easy to explain. The method of checking the sector of the map with the most unvisited coordinates leaves the navigation up to the robot, which uses its knowledge gathered by SLAM to decide the best path to the next waypoint. Sometimes the robot is given a coordinate that is outside the bounds of the world it inhabits, or requires it to take a complicated path to its goal. This is most apparent in mazeWorld, where the robot has to backtrack quite a lot in order to reach its desired location. There were occasions where the robot had to stop for several minutes for the timeout to kick in and give it a new waypoint to navigate to.

Other than these issues, the algorithm written by this team works well for the main maps provided, and still provides a good effort on the more difficult mazeWorld map. The results attained by the team are the product of many hours of hard work, and all involved are thrilled with the results.