RTAB-Mapping

Forrest Edwards

Abstract—A virtual mobile robotic platform is tele-operated in a simulated environement to perform mappng using the RTAB-Map algorithm. The robot is a 6 wheel design equiped with a Kinect RGBD camera and a Hokuyo LIDAR. Control of the robot, processing the sensor data and producing the maps is performed using ROS. Environments are modeled in the Gazebo physics simulation application and are representative of indoor residential or commercial spaces. In both cases, the robot was able to recover occupancy grids and point clouds that accurately represent the environments.

Index Terms—Robot, IEEEtran, Udacity, LaTeX, ROS, SLAM, RTAB-Map, Navigation, Machine Vision.

1 Introduction

For a mobile robot to purposefully navigate an environment it must have a map to do so. Robots can of course navigate randomly without a map but maps provide multiple advantages over random navigation that may not be initially obvious. For example, how would a robot know whether or not it has visited the same location before without a map? If an object is blocking its way, what is the best path around it? Now that the robot has figured out the best path around, does it remember that for next time? How would the robot know if it is moving in a straight line, or just traveling in large circles? How would the robot return to its starting point where, perhaps, it's charging base is located? The challenges presented by these questions are fundamental to robotics since many of the use cases for autonomous robots apply to environments where the map is unknown. Extra-terrestrial bodies, undersea trenches and living rooms are just a few examples of un-mapped environments. SLAM (Simultaneous Location and Mapping) methodologies represent a family of algorithms that are designed to address this challenge.

The work presented here will focus on one particular SLAM algorith, graph-SLAM. The mapping portion will be the primary focus leveraging the RTAB (Real Time Appearance Based) graph slam algorithm. This SLAM variant uses a combination of odometry, laser range-finder, and rgbd camera data to create both 2D and 3D occupancy grids (aka maps) in unknown environments.

2 Background

Given a map, a goal and a localization solution such as Adaptive Monte Carlo Localization [1] [2] robot navigation becomes a problem of choosing the best path and following it. But if the location has not been mapped, and mapping isn't feasible, a more generalized approach is required.

As a first proposal one might postulate a set of all probable maps for an environment and, based on the robots movement and feedback from sensors, choose the map with the highest statistical likelihood of being the correct map. While simple to conceive, this approach does not work outside of trivial cases. The problem being it's high dimensionality since the cardinality of the map set scales exponentially with the number of grid points:

$$|Maps| = 2^{|gridpoints|}$$

One way to overcome this challenge is to address it not from a map level perspective, but instead from a gridpoint perspective. Since each gridpoint has only 2 possible states (occupied, unoccupied) the high dimensionality issue is resolved. This can be represented by the factorization over the all the individual grid points, m_i, up to time t as follows:

$$\prod_i P(m_i|z_{i:t}, x_{1:t})$$

This approach overcomes the dimensionality challenge by predicting which gridpoints are occupied but is still insufficient to be considered a SLAM solution because it can not answer the question "have I been here before?" To solve this challenge, a process for matching a prior pose and observations, with current pose and observations is needed.

Correspondence is the term used to describe the likelihood that two observed environments are the same and is a critical piece of the mapping task. Odometry can, in part, be leveraged but even the best sensor data has noise. In situations where the global environment is large or the mapping duration long the accumulated odometry error eventually becomes unusable for correspondence purposes.

Multiple solutions to the correspondence problem have been implemented successfully. One such solution, the EKF-SLAM algorithm, extracts landmark features as the robot moves and represents them as Gaussian distributions and covariances. However, it suffers from unconstrained memory usage as the number of landmarks observed grows. Additionally the Guassian landmarks are necessarily low fidelity which can result in excessive false correspondence if the confidence in odometry is low.

FastSLAM is another approach that uses custom particle filters, each containing the robot trajectory and a map of features represented by local Gaussians. Feature correspondence is handled similarly to the approach used by EKF-SLAM [3], but for each particle. This modified approach of utilizing particle filters and Gaussians is known as the Rao-Blackwellized particle filter approach [4]. Three different versions of the FastSLAM algorithm exist but Grid-Based

has the advantage of not requiring known correspondences. One drawback of FastSLAM as with all stochastic particle filters is that with a finite number of particles it is unlikely that any particle will perfectly match the robots most likely position so it always exhibits some inherent error.

GraphSLAM [5], the solution selected for this work, departs from the previously discussed approaches in that it solves the full SLAM problem rather than just the most recent pose and map. This allows it to recover the full map and path. GraphSLAM is founded on the concept that a robot's motions and measurements can be modeled as constraints that influence the probability of a robot's pose. The constraints (more accurately *soft constraints* due to their uncertainty) are graphically represented as shown in Figure 1. As can be seen multiple measurements can be defined to a landmark for different poses. Once modeled, the goal of the GraphSLAM algorithm becomes optimizing the poses and constraints to determine the map with the highest likelihood. In practice this optimization is done numerically vs. analytically and is typically iterative to accommodate non-linear constraints.

The version of GraphSLAM leveraged for the work here is the RTAB-Map variant. RTAB(Real-time appearance based) mapping uses camera image data to establish correspondence between observations. It is suited for large scale, long term, and multi-session SLAM tasks. RTAB-Map does not rely on defined landmarks (known correspondence); instead a bag of words approach is used to establish correspondence in near real time.

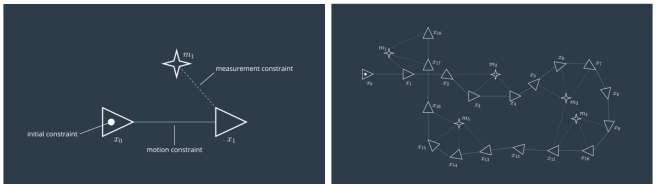


Figure 1. Sample GraphSLAM graphs. *Triangles: Poses, Stars:Landmarks, Solid Lines: Motions, Dashed Lines: Measurements*

The main component in RTAB-Map's visual approach is implementation of the Visual Bag of Words algorithm. The Bag of Words algorithm leverages the SURF (Speeded Up Robust Features) [6] algorithm to extract features from images. These features are used to build a vocabulary of visual words that describe each image as shown in Figure 2. When the visual word vectors for two images reach

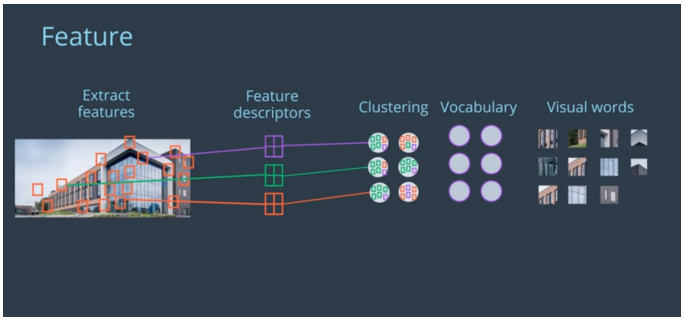


Figure 2. visual representation of Bag of Words feature extraction process.

a defined level of similarity, the algorithm creates a loop closure constraint associating the two poses relative to the observed landmark. Since loop-closure requires that visual descriptions be maintained for all previous observations RTAB-map also implements robust controls to cap memory and CPU utilization. This allows RTAB-map suitable for a wide variety of platforms, including resource constrained embedded systems.

Since the robot modeled here is a surface bound six wheeled design the mapping task can be simplified to a 2D mapping problem with minimal loss of fidelity for navigation. However, RTAB-Map is also suited for 3D mapping as, for example, would be needed for a quadcopter SLAM implementation. Of course, with the addition of the thrird dimension memory required for mapping sscales with the cubic and thus memory management becomes much more critical for implementation on hardware constrained platforms. Octomap [7] is one potential solution that has advantages over point cloud and trivial voxel based approaches. Octomap supports probabilistic occupancy estimation similar to the 2D occupancy grid leveraging a recursive binary Bayes filter and log odds notation. It also supports the representation of unmapped voxels in addition to the binary occupied/ unoccupied states represented by trivial point clouds. Lastly, Octomap includes a framework to optimize memory by coalescing similar adjacent child voxels into larger,more memory efficient parent voxels. This is illustrated in Figure 3, which also gives some insight into the genesis of the 'Oct' in Octotree and Octomap.

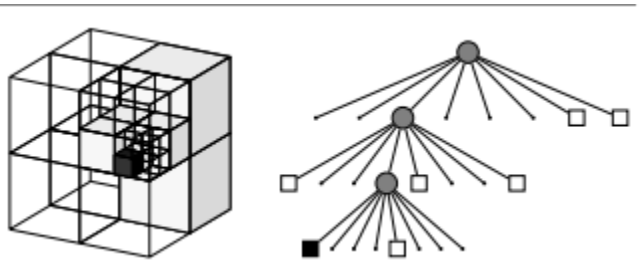


Figure 3. Illustration of Octomamp voxel parent-child heirarchy. [7]

3 Model Configuration

The custom robot built previously was used for as the platform for implementing the mapping work. It's wide flat surfaces were designed with the intent of accomodating additional hardware, and in this respect the Kinect RGBD camera integrated seamlessly.

Two configurations for the Kinect were implemented as can be seen in Figure 4 The first mounted the Kinect inverted, which was the most straightforward solution from an attachment perspective. However, while this configuration provided correct point cloud orientations, RGB images were inverted. The second configuration attached the Kinect in an upright orientation to compare impact of inverted and upright RGB images on the RTAB algorithm. An ideal solution would have been to modify the optical frames for the camera and RGBD point clouds independently, but there does not currently appear to be a solution to do this and

rewriting the Gazebo Kinect plugin is beyond the scope of this work. As mentioned above, an optical frame joint

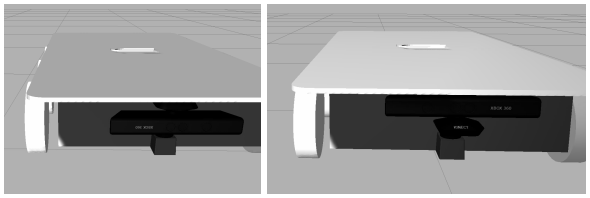


Figure 4. Placement of kinect on robot base. Left: inverted orientation, Right: Upright orientation.

was added between the robot and the Kinect to correct for differences in choice of origin pose between OpenCV (used to render point clouds from the Kinect) and Gazebo. Without this, camera point cloud data was displayed above the robot, rather than in front of it.

In addition to the physical configuration changes, the ROS configuration was also modified to fully integrate the RTAB-Map and teleop nodes into the robot. RTAB-Map required a correction to the Grid/CellSize parameter to prevent it from crashing on launch due to a known issue that results in a failed assertion.Through trial and error it was found that a CellSize of .06 (vs. the default of .05) would function, but a value of .075 was selected to deliver more logical grid spacing. The frameID and scan topics were also remapped to match with the robot's existing topics and tf frame names.

The teleop cmd_vel topic required remapping to the global name space to function properly which is notable as the currently maintained official ROS teleop node publishes to the global name space. However, the provided node was modified to publish in the teleop name space.

An overview of the launch files in which these updates were made, their interconnections and actions is included in the appendix as Figure 8. As can be seen there, the package includes 3 primary launch files. The RVIZ launch file was omitted and integrated into the main launch file as, contrary to the mapping and teleop nodes, there was not a compelling reason to have a dedicated RVIZ terminal open.

4 World Creation

Two worlds were mapped as part of this work. The first was a world provided by Udacity representing an kitchen and dining room. For the second world, creative direction was left to the student. That world takes the form of a larger space representing an office break room. In both cases, the worlds were not built from scratch but assembled from existing models found in the Gaezebo model database [8].

4.1 Kitchen Dining World

The Kitchen Dining world shown in Figure 5 was used for the majority of testing and mapping performed for the project. The primary challenge encountered in implementing mapping in this world resulted from the collisions not initially being enabled in the world. Since the Hokuyo LIDAR gazebo plugin relies on collisions being enabled, mapping could not proceed until the issue was addressed. Using the collision enabled model set from section 12 of the

Figure 5. Top view of virtual environment used for mapping.

lectures, a new kitchen dining world file was created with collisions enabled and used for this project.

4.2 Break Room World

The break room model shown in Figure 6 was created in Gazebo. The bulk of the architectural features are part of an existing model, however multiple features were added to visually enrich the environment including:

- Cafe Table
- Control Console
- Picture (Checkerboard)
- Picture (Spectrum Plane)
- Oak Tree
- Pine Tree
- Cabinet
- Wall Mounted bookskelf
- Marble Table
- Husky Robot
- Wooden Crate

Figure 6. Persepective view of Break Room world created for project.

While aesthetically pleasing, these additions serve a more functional purpose in the world. Without these additions, the large monolithic surfaces present in this environment present little to no variation and the RTAB algorithm would fail to identify consistent loop-closure constraints.The images in Figure illustrate this challenge. The two lower images are practically identical, and even a human observer would be challenged to identify which location belonged to which image. This is exacerbated in virtual environments since textures, like the floor, repeat identically throughout the environment with a high level of similarity

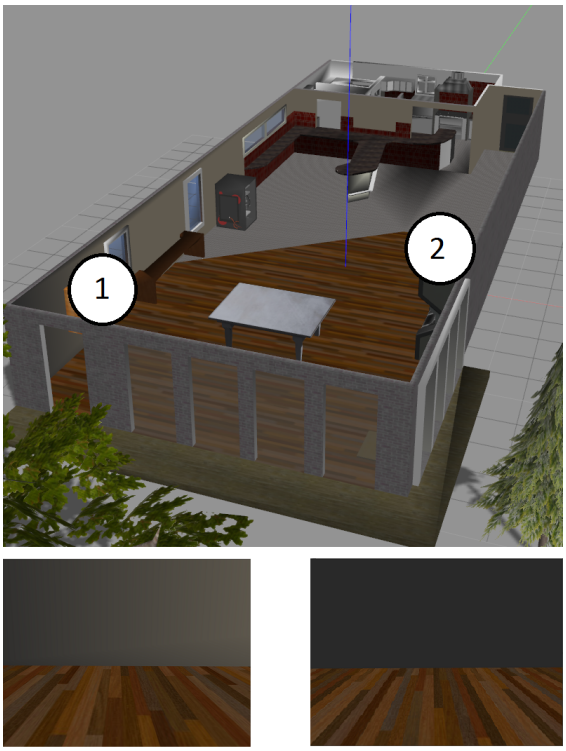


Figure 7. Illustration of the kind of similarity exhibited in the model without the addition of additional features. The left image was captured in location 1, while the right image was captured in location 2

5 Results

Multiple mapping runs were made of the Kitchen Dining world. In each run 3 passes through the environment along approximatetly the same path were made. The goal being to evaluate the effect of tuning parameters on closure and mapping performance. Additionally, an additional run was made in the default configuration with the Kinect oriented upright to understand the impact if any. For the created Break Room world, a single run was made with the default setup. Results from the runs are tabulated in the appendix as Table 1 and Table 2.

In each case the robot was able to build a recognizable occupancy grid and 3D point cloud, although the level of accuracy vary from case to case. Additionally the number of loop closures for each run was significantly above the project required minimum of 3 with the greatest amount seen on run 5 (435), and the least amount seen in the Break Room run(52).

6 Discussion

Several notable observations came out of the results presented here. Firstly, in multiple runs (1,1b,2,3) the lower wall of the occupancy grid exibits a fanning effect. By reviewing the closure constraints in rtabmap-database viewer it appears that little to no closures were made in this region as the robot traveled from left to right alongside the featureless wall. One possible explanation is that multiple runs in feature sparse areas can not be spatially converged due to the lack of loop closures. This idea seems to be supported by the Break Room run as well, where along the top wall, where the checkerboard and spectrum paintings are hung, the occupied grid cells are spatially converged, however as one moves down the wall the occupied grid points begin to fan out in the featureless region.

It's also surprising that for these environments, the orientation of the Kinect appears to have had little effect on the quality of the occupancy grids produced. It is possible this results from the scale and rotation invariance of the SURF algorithm in detecting features in the images.

As alluded to previously, the difference in RTAB-Map performance between worlds appears to be direclty related to the feature density present. In areas where there are many distinct features to make loop closures, the accuracy is the highest. Conversely, in feature poor areas, the accuracy of the grid degrades as the distance from the nearest loop closure increases.

7 Future Work

This work could be enhanced significantly by implementing and testing it on a physical robot.There is also an opportunity to further refine and build out testing of the various parameters to determine how much improvement can be gained by further tuning the mapping parameters.

References

[1] D. Fox, W. Burgard, F. Dellaert, and S. Thrun, "Monte Carlo Localization: Efcient Position Estimation for Mobile Robots," p. 7.
[2] S. Thrun, D. Fox, W. Burgard, and F. Dellaert, "Robust Monte Carlo localization for mobile robots," *Artificial Intelligence*, vol. 128, pp. 99–141, May 2001.
[3] M. Montemerlo and D. Koller, "FastSLAM: A Factored Solution to the Simultaneous Localization and Mapping Problem," p. 6.
[4] K. Chamaa, "Robotics Software Engineer - Udacity."
[5] S. Thrun and M. Montemerlo, "The Graph SLAM Algorithm with Applications to Large-Scale Mapping of Urban Structures," *The International Journal of Robotics Research*, vol. 25, pp. 403–429, May 2006.
[6] H. Bay, A. Ess, T. Tuytelaars, and L. V. Gool, "Speeded-Up Robust Features (SURF)," p. 14.
[7] A. Hornung, K. M. Wurm, M. Bennewitz, C. Stachniss, and W. Burgard, "OctoMap: an efficient probabilistic 3d mapping framework based on octrees," *Autonomous Robots*, vol. 34, pp. 189–206, Apr. 2013.
[8] "osrf, Gazeebo Model Database."

Appendix

Model & World Modifications

1) Add optical joint to correctly orient Kinect Point Cloud.
2) Added Kinect sensor to optical joint.
3) Removed all config files from config directory.
4) Removed launch files from launch directory.
5) Created teleop, kitchen_dining, mapping and breakroom launch files.
6) Delete existing jackyl map and world files.
7) Placed kitchen_dining and breakroom world files into worlds directory
8) Fixed issue with kitchen_dining not having collisions enabled.
9) remap /teleop/cmd_vel to /cmd_vel
10) Grid/CellSize adjusted to 0.075
11) updated frame_id value in mapping.launch to robot_footprint

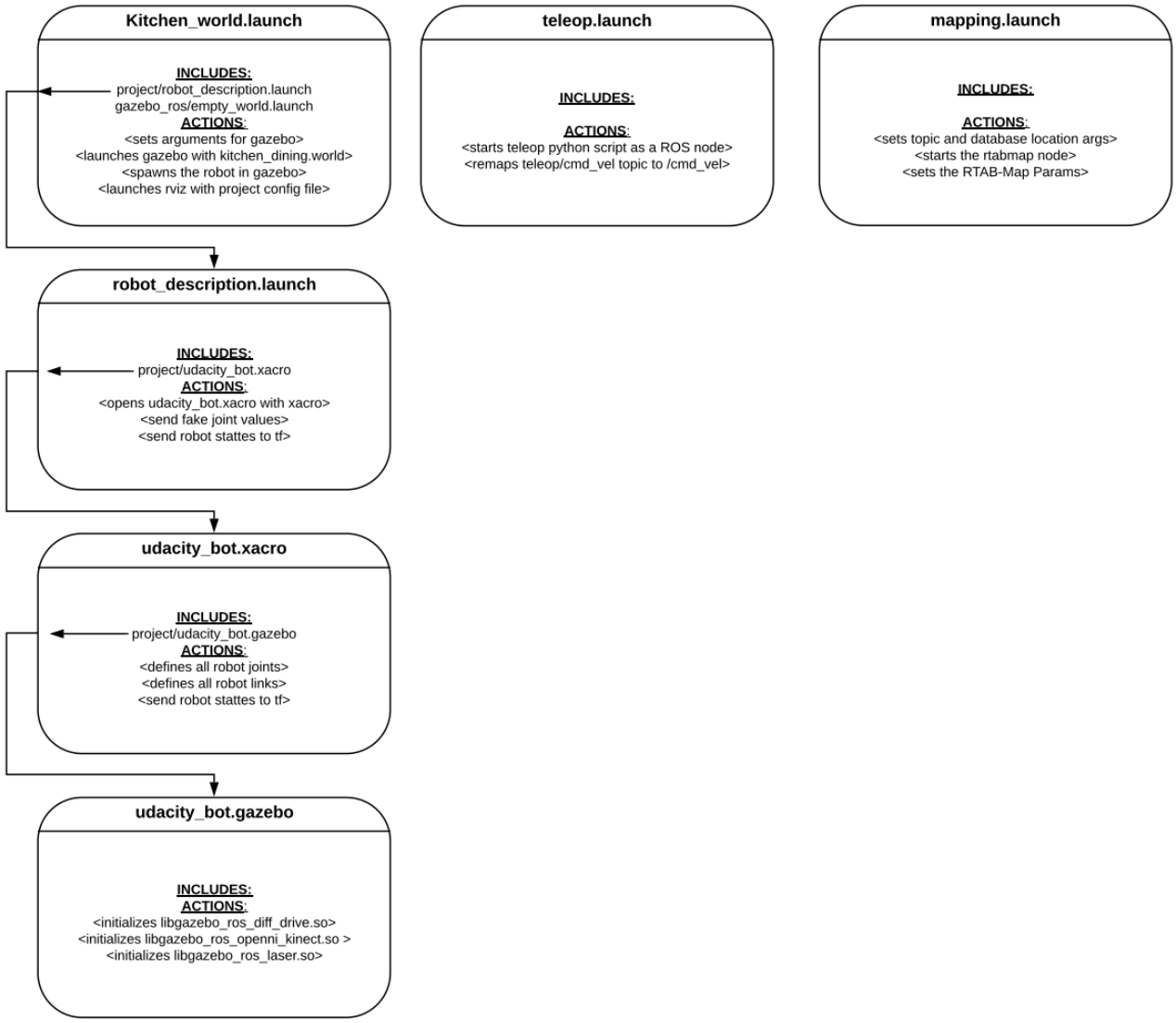
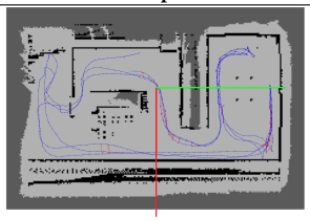
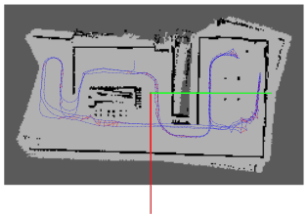
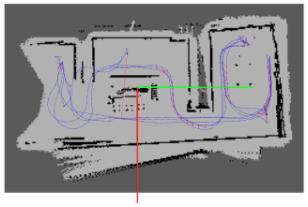
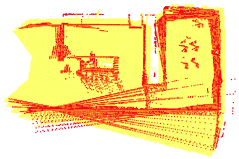
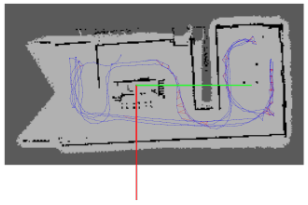
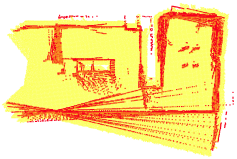
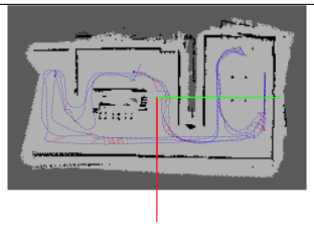
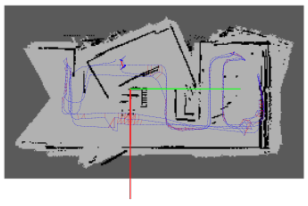


Figure 8. Visual depiction of launch files and their relationships.

Table 1
Summary of parameter adjustments and Loop closures for each run

Run	Kinect	Param	Value(s)	Global Closures
1a	Inverted	Defaults	Defaults	222
1b	Upright	Defaults	Defaults	390
2	Inverted	Odom Strategy	Frame to Frame	231
3	Inverted	Vis/Cortype	Optical Flow	158
4	Inverted	OdomF2M/MaxSize Vis/MaxFeatures	1000 600	311
5	Inverted	Optimizer/Slam2D	true	435
Breakroom	Upright	Defaults	Defaults	52

Table 2
Graph, occupancy grid, and point cloud visualizations for each run

Run	Graph	Occupancy Grid	Point Cloud
Run 1a			
Run 1b			
Run 2			
Run 3			
Run 4			
Run 5			
Breakroom