eYSIP2017

# Indoor Environment Mapping Using UAV



Rishabh Beri
Samaahita S Belavadi
Mentors: Simranjeet, Vamshi
Duration of Internship: 22/05/2017 – 07/07/2017

*2017, e-Yantra Publication*

# Indoor Environment Mapping Using UAV


## Abstract

Research in the area of robotic indoor mapping has been active for a very long time. This project aims to solve the problem of autonomous mapping of an indoor environment using an unmanned aerial vehicle. The platform chosen to solve this problem is Robotics Operating System(ROS) and Gazebo simulator.
ROS is an open-source system for controlling robots via PC. It has various libraries and packages that simplify the task of building robotic software applications.
Gazebo is a simulator powered with a physics engine, used to simulate the real world. It enables setting of parameters like gravity, friction, inertia, etc. to replicate any real world scenario.
The package used for map building is RTAB-Map which runs an RGB-D Graph SLAM algorithm based on the global Bayesian loop closure detector. In simulation, this problem was solved using an Intel Realsense R200 camera mounted on an AR Drone 2.0. In the real world, we used a Kinect v1.0 camera mounted on a Firebird VI robot, which is controlled manually.


### Completion status

- Completed autonomous mapping in simulation
- Interfaced Kinect camera with the PC
- Completed manual mapping in the real world
- Made tutorials on the various packages used in the project
- Video explaining the autonomous mapping algorithm used

## 1.1 Hardware Components

- Kinect v1.0 (Specifications)
- Firebird VI (Datasheets and Manuals, Buy here)
- USB to RS-232 Converter (Datasheet, Buy here)

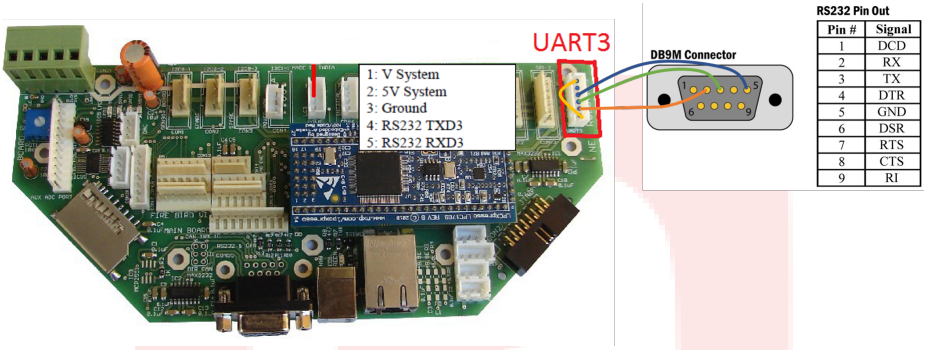


Figure 1.1: Circuit Diagram

## 1.2 Software used

- Ubuntu
  - Version - 14.04
- ROS
  - Version - Indigo
  - Installation:
    `sudo apt-get install ros-indigo-desktop-full`
- Gazebo Simulator
  - Version - 7
  - Installation - Download and run this file to install gazebo7

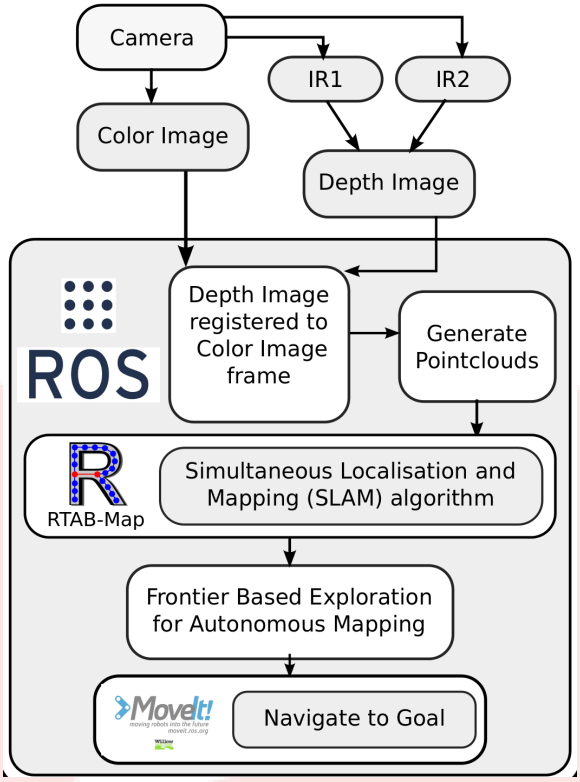


Figure 1.2: The mapping process

## 1.3 The Mapping Process

### Packages Required

- ardrone_simulator_gazebo7 - This package contains the models, drivers and plug-ins necessary to simulate the behaviour of an AR Drone in Gazebo7.
- depth_image_proc - This package is a part of the image pipeline package of ROS, and contains nodelets for processing depth images.
- realsense_gazebo_plugin - This package contains the models, drivers and plugins necessary to simulate the behaviour of an Intel Realsense R200 camera in Gazebo7.
- move_it - This package plans the path of the drone from it's current location to a goal point.

## Initial Setup

- Including the drone and camera in gazebo
  - The model of the AR Drone is in the URDF format
  - But, the model of the Realsense camera is in SDF format
  - URDF and SDF formats are incompatible. Hence, one format has to be converted to the other
  - This tutorial explains the conversion process

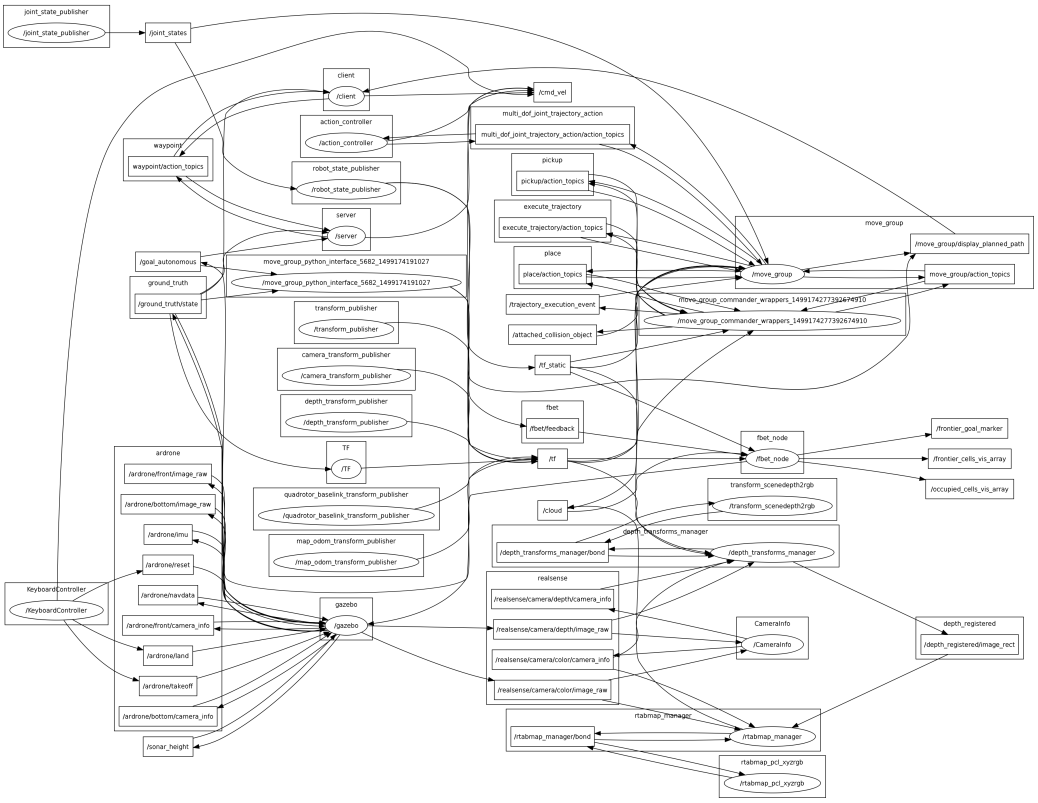


Figure 1.3: Output of the rqt_graph command showing how the nodes communicate with each other

## Generation of Pointclouds

- Register (i.e. project) the depth image to the frame of the colour image
  - The realsense camera publishes RGB and depth images on topics /realsense/camera/color/image_raw and /realsense/camera/depth/image_raw respectively
  - But to generate an RGB-D pointcloud, the depth image and the colour image have to be in the same frame.
  - The register nodelet (i.e., depth_registered in rqt_graph) of the depth_image_proc package is used for registering the depth image to the frame of the colour image
  - Requirements:
    - Meta-data of the RGB camera (Published on the topic /realsense/camera/color/camera_info)
    - Meta-data of the depth camera (Published on the topic /realsense/camera/depth/camera_info)
    - Depth Image (Published on the topic /realsense/camera/depth/image_raw)
    - A transform between the frame of the depth camera and that of the colour camera (A static transform between color and depth published on the topic /tf)
  - The output of this nodelet is a registered depth image (Published on the topic /depth_registered/image_rect)
- Create a pointcloud
  - The point_cloud_xyzrgb nodelet (i.e., rtabmap_pcl_xyzrgb in the rqt_graph) of the rtabmap_ros package is used to generate a pointcloud in which each point has 7 attributes - 3D co-ordinates, RGB-colour and depth.
  - Requirements:
    - RGB image (Published on the topic /realsense/camera/color/image_raw)
    - Registered depth image (Published on the topic /depth_registered/image_rect)
    - Meta-data of the RGB camera (Published on the topic /realsense/camera/color/camera_info)
  - The output of this nodelet is a pointcloud (Published on the topic /cloud)

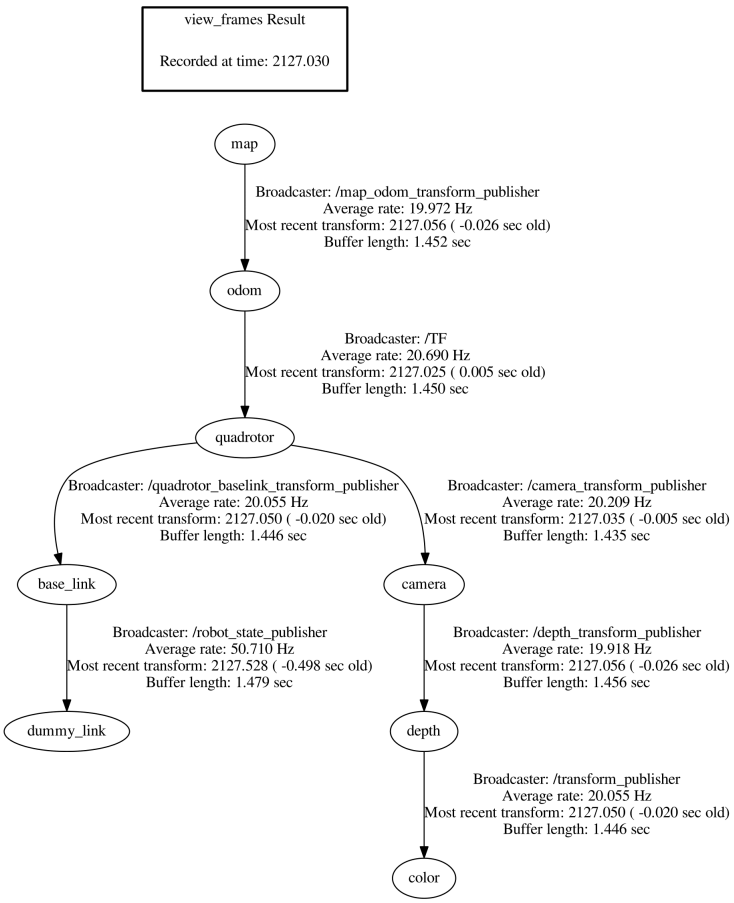


Figure 1.4: The tf tree generated during the mapping process

## Map Building using RTAB-Map

- SLAM - Simultaneous Localization and Mapping
  Simultaneous Localization and Mapping (SLAM), is the problem of building the map of an unknown environment incrementally, while simultaneously keeping track of a robot's location within the map.
  The SLAM algorithm applied depends on the sensors used in the mapping process. These algorithms are broadly classified into landmark-based and raw-data algorithms. Landmark-based algorithms uniquely identify objects in the world, whose location is estimated using a sensor. Raw-data algorithms on the other hand, model the probability of estimating the correctness of an observation directly as a function of

the location.
RTAB-Map is a RGB-D Graph SLAM approach based on the global Bayesian loop closure detector. Loop-Closure is the problem of identifying a previously visited location based on sensed features and updating the variables accordingly. The loop closure detector uses a bag-of-words approach to determinate how likely a new image comes from a previous location or a new location.

- The main node of the RTAB-Map package is rtabmap
- The map is incrementally built and optimized whenever a loop-closure is detected
- Requirements:
  - Odometry information (Published on the topic /ground_truth/state)
  - RGB image (Published on the topic /realsense/camera/color/image_raw)
  - Metadata of the RGB image (Published on the topic /realsense/-camera/color/camera_info)
  - Registered depth image (Published on the topic /depth_registered/image_rect)
  - Pointclouds (Published on the topic /cloud)
- The output of this node is an incrementally built map and information about the map
- There are various parameters that can be set to use the node for a wide variety of applications (More information about these parameters can be found here)
- This tutorial explains how to use RTAB-Map for mapping

## Autonomous Mapping - Frontier-Based Exploration Technique (FBET)

- Algorithm for Goal generation:
  1. Receive a Pointcloud (Published on the topic /cloud)
  2. Convert the Pointcloud to an Octomap
  3. Classify cells in the Octomap as free or occupied
  4. Find frontier cells

   - Frontier cells represent the boundary of known and unknown regions
   - A free cell is a frontier cell if it has one free cell and one unknown cell as it's neighbour
5. Cluster the frontier cells using the k-means clustering algorithm
   - The number of clusters 'k', depends upon the number of frontier cells
   - Since the camera has a limited field of view, we limit the number of frontier cells each cluster can accommodate
6. Calculate the cost of each cluster. The cost function depends on:
   - Inverse of the cluster size - Cluster size represents the size of the unknown region
   - Distance between the drone and the centroid of the cluster
7. The cluster with the minimum cost is chosen the optimal cluster
8. The centroid of the optimal cluster is the new goal to navigate to (This new goal is published on the topic /goal_autonomous)

- Path planning
  - Package used - MoveIt
  - MoveIt is a ROS framework used for motion planning, control and navigation of robotic arms
  - By making a few changes, we can use it for planning the path of a quadrotor too
  - If it is not possible for MoveIt to generate a path to a goal(Goal is too close to an obstacle), then a feedback mechanism requests the autonomous_exploration node for a new goal (The feedback is sent by sending an empty message to the autonomous_exploration node over the topic /fbet/feedback whenever MoveIt is unable to generate a plan)

- Navigating to the goal
  - Once a plan has been generated, we use actionlib server-client mechanism along with PID control to navigate through each waypoint of the goal (These waypoints are extracted from the topic /move_group/display_planned_path)

- This video explains the mapping process

## 1.4 Software and Code

Github link for the repository of code

- /ardrone_simulator_gazebo7/cvg_sim_gazebo/scripts/aruco_land.py - Script to execute landing of the drone on an ArUco marker.
- /ardrone_simulator_gazebo7/cvg_sim_gazebo/scripts/keyboard.py - Script to control the drone manually
- /ardrone_simulator_gazebo7/cvg_sim_gazebo/scripts/ardrone_get_odometry.py - Script to fetch pose of the drone from gazebo and publish tf
- /realsense_gazebo_plugin/scripts/pub_camera_info.py - Publish fake depth and color camera info for the simulated cameras
- /realsense_gazebo_plugin/scripts/server.py - Start the actionlib server to execute the waypoints from MoveIt!
- /realsense_gazebo_plugin/scripts/client.py - Start the actionlib client to execute the waypoints from MoveIt!
- /realsense_gazebo_plugin/scripts/send_goal.py - Start the script to send goals to MoveIt!
- /realsense_gazebo_plugin/launch/ardrone_realsense.launch - Launch Gazebo7 and load the simulated world
- /realsense_gazebo_plugin/launch/register.launch - Register the depth image to the color image frame
- /realsense_gazebo_plugin/launch/rtabmap_pcl.launch - Launch nodelet to convert depth and color image to pointcloud
- /realsense_gazebo_plugin/launch/fbet.launch - Launch the autonomous_exploration node which runs the FBET algorithm
- /move_it/launch/moveit.launch - Launch the MoveIt! path planner
- /autonomous_exploration/src/autonomous_exploration.cpp - Code to find goals for autonomous navigation
- /bash_scripts/keyboard_mapping.sh - Bash script to launch all necessary packages required for the mapping process

## 1.5 Tutorials

- Adding Realsense R200 camera to ARDrone in Gazebo7
- Landing Drone on ArUco marker
- OctoMap and RTAB-Map
- Using RTAB-Map for mapping with Kinect
- Keyboard Mapping
- Autonomous Mapping

## 1.6 Results and Demo

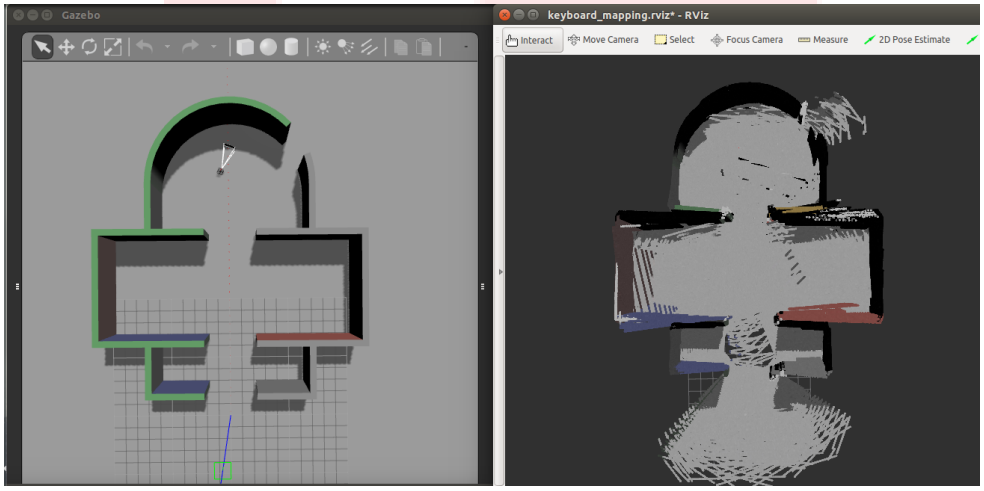


Figure 1.5: Result of manual mapping in simulation

Demonstration video for manual mapping in simulation

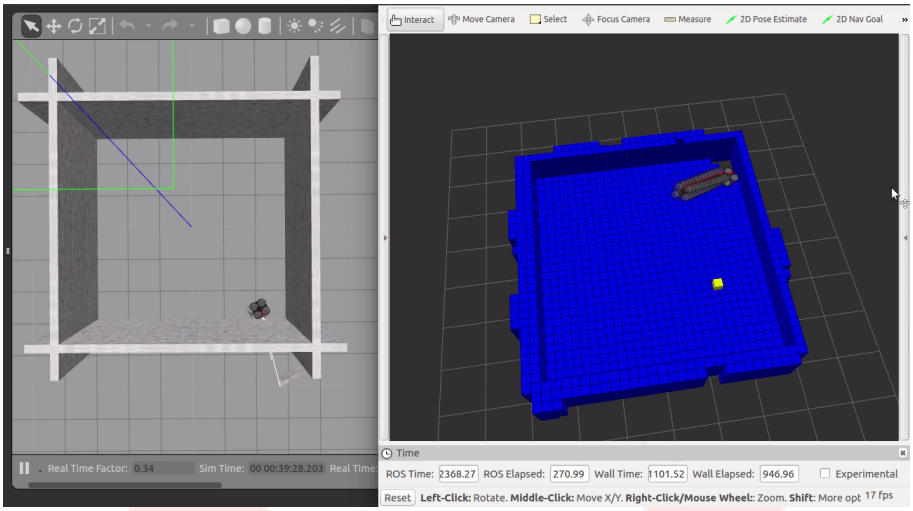


Figure 1.6: Result of autonomous mapping in simulation

Demonstration video for autonomous mapping in simulation


Figure 1.7: Final setup

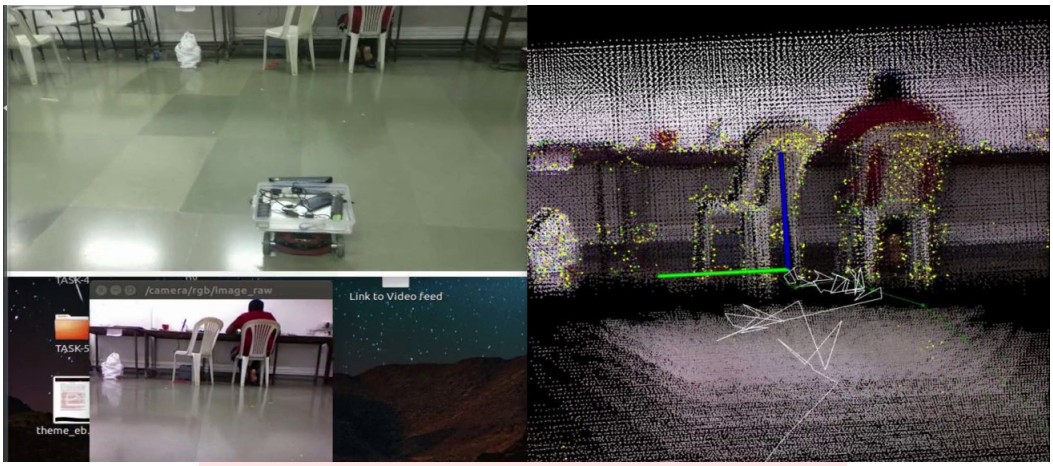


Figure 1.8: Result of manual mapping in the real world

Demonstration video for manual mapping of the real world

## 1.7 Future Work

- Autonomous mapping of the real world using a Firebird VI robot or a quadcopter.
- Build a system comprising of two drones/robots where one of them maps the environment autonomously, while the other navigates autonomously in the generated map to perform a simple task.

## 1.8 Bug report and Challenges

- Issues:
  - The mass of the Realsense R200 camera was set to 1 gram in Gazebo. Setting it to a higher value increases the load on the drone, which causes a constant decrease in altitude.
  - When the drone goes very close to an obstacle in Gazebo, the camera is able to see through the obstacle and hence pointclouds which would be impossible to generate in the real world are generated in the simulated world.
  - Due to the numerous front-end and back-end processes running, the processor(i5 5th Gen Dual-Core) was overloaded. This created

a huge delay between the generation of pointclouds and the map building process.

- The MoveIt package was developed for robotic arms. Hence, when using the package for a drone, the planned trajectories are overly complex.

- Failures:
  - Due to compatibility issues with the RGB-D camera and the processor, the original idea of mapping with a drone in the real world had to be dropped. Hence, we used a Firebird VI robot for manual mapping.

- Challenges faced:

1. Problem: ROS comes pre-installed with Gazebo2, but the Realsense plugin works only with Gazebo7 and higher.
   Solution: Replaced Gazebo2 with Gazebo7.
2. Setting the right values for the PID parameters
3. Problem: The Gazebo plug-in for Realsense camera does not publish camera metadata and calibration parameters(camera info)
   Solution: Wrote a python script to publish camera info (obtained from the physical realsense camera) for depth and color cameras in sync with the image data
4. Problem: The default encoding of the depth image is mono16, which is not supported by the depth_image_proc nodelets
   Solution: Changed the encoding of the depth image from mono16 to 16UC1 using the toCvShare function of the cv_bridge library
5. Problem: Register nodelet of depth_image_proc requires a transform from the depth image to the color image
   Solution: Created a static_transform_publisher node to publish tf data (based on placement of the depth and color cameras in the SDF model of the realsense camera)
6. Problem: Choosing between the octomap package and rtabmap package of ROS to build a map
   Solution: Octomaps require a localization process(e.g. hector_mapping) running in the background, whereas RTAB-Map runs a SLAM algorithm based on loop closure detection for localization and mapping. Hence we chose to use RTAB-Map to create a map of the environment

7. Problem: Choosing an efficient algorithm for autonomous mapping
   Solution: Two options were available - Wall following, and frontier-based exploration. We chose frontier-based exploration because, the wall following strategy was limited to only rectangular walls and does not perform well with obstacles in the map.
8. Problem: Choosing a good clustering technique for the frontier-based exploration algorithm
   Solution: We chose the k-means clustering algorithm to cluster the frontier cells, because it is one of the most efficient clustering algorithms for big data sets and the clusters generated are of similar sizes
9. Problem: Choosing an optimal value of K in the k-means clustering algorithm
   Solution: The efficiency of the entire algorithm depends on the value of K (number of clusters). A very low value of K would give us large clusters which would be useless given that the drone has a limited field of view. A large value of K would give us very small clusters, which would slow down the mapping process. Hence, we fixed an upper limit on the number of frontier cells each cluster can accommodate, and found the value of K.

# Bibliography


[1] Labbe, M. and Michaud, F., *Online Global Loop Closure Detection for Large-Scale Multi-Session Graph-Based SLAM*, 2014

[2] Labbe, M. and Michaud, F., *Appearance-Based Loop Closure Detection for Online Large-Scale and Long-Term Operation*, 2013

[3] Cheng Zhu, Rong Ding, Mengxiang Lin, Yuanyuan Wu, *A 3D frontier-based exploration tool for MAVs*, 2015

[4] ROS Wiki

[5] Gazebo simulator

[6] RTAB-Map

[7] MoveIt!

[8] MoveIt for quadrotors

[9] RS232 Connector