Map My World Robot

Humberto MartinezBarron

Abstract—A ROS package was built to perform Simultaneous Localization And Mapping (SLAM) of a given and a custom environment by interfacing with the Real-Time Appearance-Based Mapping (RTAB-Map) package. This project has been developed with the building of a rescue robot in mind. Although there might still be several limitations, the mapping performed by the robot is complete and easily understood, meaning this solution might be well-suited for deployment in rescue missions during natural disasters in a later, more sophisticated version.

Index Terms—SLAM, loop closure, bag-of-words, caster, link, joint, RGBD camera, hokuyo, odometry data, sensory data, noise, correspondence, measurement constraint, motion constraint, poses, features, RTAB-Map.

1 Introduction

The problem this project is trying to solve is known widely in the robotics field as SLAM (Simultaneous Localization And Mapping). Before a solution to this problem came along, robots needed to be provided a map to perform localization (or the robot's location to perform mapping) and thus begin to carry out very basic tasks. However, every time the environment changed or the robot was moved to a new location, a new map needed to be fed to the robot in order for it to be able to continue going about doing its job. This might be just fine in some cases - like robots that only move in a limited space that does not change often. However, it was impossible to achieve futuristic, science-fiction-like robots with this approach (just imagine what would happen if R2D2 and C3PO were unable to move around in environments they had never seen before!).

In comes SLAM. This approach has a simple idea behind it, pretty much saying: "hey, we've got odometry (wheel) data that we can use to figure out where the robot might be with respect to its initial position, and we have sensory data that can sense how far away objects in the environment are, and how they look. Can't we put all of that together and get R2 to move in any new space?"

This is a very smart approach, and the answers found to this question are fascinating - indeed, it is possible to combine odometry and sensory data (as in pure localization) to figure out where the robot might be with respect to its original position and where the objects it is looking at with its sensors might be, their shape, color, and maybe even their texture. However (spoiler alert), this approach is haunted by the worst enemy of roboticists: NOISE. "Noise" is not just that thing Skrillex makes, it's also - in the world of robotics - errors in measurements due to imprecise sensors. There are no perfect sensors - if there were, most roboticists might be out of a job! Therefore, the answers to this question have to be developed further in order to solve the SLAM problem.

NOTE: in order to better follow with this paper, it is recommended to download the project, named slam_project.

2 Background

2.1 Why is SLAM important?

Okay, let's dive into what SLAM does in order to properly generate a map! But first, let's take a step back and ask this question: is it even worth it? Every time before starting to try solving a problem, it is a good idea to ask if the problem is worth solving in the first place (spoiler alert again, it is, but let's see why).

Mapping allows robots to generate maps (duh)! And what would happen if they could not generate maps? Well, going back to our R2D2 and C3PO example, no map would mean

they would be stuck in a room all day because they could only operate in locations with given maps, and these locations would need to stay almost exactly the same all the time! So they could only work at a factory, or maybe a house as service robots - and that would make not only for a very dull movie, but for a pretty uncool robot.

2.2 The Idea

Moving to an example limited to this galaxy, if a robot sent on a rescue and reconnaissance mission in the middle of rubble left by an earthquake in places too narrow or too dangerous for humans could not generate a map of its surroundings and localize itself simultaneously, then the poor little robot would get lost, and now we would have to launch a double rescue mission! That is obviously not the objective of this project, so yes, the mapping problem is definitely worth solving at the same time as the localization problem.

2.3 Solving the Problem

Now for the fun part - how does SLAM work? And why is it so hard? Well, SLAM is such a big deal because it combines two already difficult problems into an even harder one. Think about it, if the map is provided but the location is unknown, then localization is feasible through an approach such as a particle filter. On the other hand, if the robot's location is provided, but the map is unknown, then sensory data can be used to put together a map of the environment. But what if the robot has no clue where it is or how the world looks like?

SLAM is a continuous-discrete estimation problem. This is only a fancy way of saying that the robot needs to figure out both its current pose (through what roboticists call Online SLAM) and its trajectory (through Full SLAM) using continuously sensing the objects around it (continuous) and by asking itself at every step: "have I been here before?" (discrete - this is called correspondence). Thus, the relationship between endogenous and exogenous variables of SLAM will be:

$$p(x_{1:t}, m, c_{1:t}|u_{1:t}, z_{1:t}) \tag{1}$$

Where $x_{1:t}$ represents the robot's trajectory (defined by its positions from moment 1 to t), m represents the generated map, $c_{1:t}$ represents the correspondence from time 1 to t, $u_{1:t}$ represents the controls (odometry) of the robot from time 1 to t, and $z_{1:t}$ represents the sensory data obtained from time 1 to t.

2.4 Mapping algorithms

Current approaches to solving the SLAM problem include:

- FastSLAM uses a particle filter with known correspondences to estimate the posterior expressed in (1), both version 1 and version 2 of this algorithm use an Extended Kalman Filter (EKF). Its extension, grid-based FastSLAM, adapts the map to a grid.
- GraphSLAM uses measurement and motion constraints to generate a (you guessed it) graph! As more constraints are added to the graph, the map becomes more complete. The constraints generated by this graph will conflict, and thus the algorithm's "magic" consists on performing optimization to find a configuration for the map that minimizes the total error. To represent this mathematically, we define an information matrix Ω and an information vector ξ that will help calculate all poses and features - expressed as a vector μ.

$$\mu = \Omega^{-1}\xi \tag{2}$$

  - RTAB-Map (Real-Time Appearance-Based Mapping) is the approach used to develop this project. The innovation of RTAB-Map is the use of loop-closures and visual bags-of-words. This means that the algorithm allows the robot to save important features (characteristics in an image such as a corner or an edge) in long-term memory to later compare them to new images it is seeing, and thus determine whether the image has been seen before, and therefore if the robot has already been there.

3 Scene and Robot Configuration

Now it is time to explain the particular solution and application proposed in this paper. Now that SLAM has been explained and the algorithm specified, it is time to move on to the world created and the focus in mind behind this project.

3.1 Gazebo World Creation

Upon spawning for the first time in the custom Gazebo world, the first thing that will appear will be a person lying on rubble. This is to bring to the attention that this robot is meant to be a rescue and reconnaissance bot. Imagine being able to rely on robots to act as first responders after natural disasters! They could access places too dangerous or too hard to access for humans (e.g. too narrow, too short, etc). This way, robots could take care of locating people who need help in the area of interest, or make a map to help determine the gravity of the damages caused in the zone without risking human lives!

In this particular example, the robot is to navigate in a collapsed city with a nuclear reactor right beside it! Of course, this is a rather unlikely scenario to say the least. However, the point is to be made that this solution is not proposed for the particular case of an earthquake in a city with a nuclear reactor in the middle of it. This exaggeration is merely used to illustrate the possibility of a natural disaster happening in a populated area and humans being unable to reach that place - plus, it looks cool (see how cool it looks in figure 6).

This Gazebo world might once have been an important city center - complete with a small park, police station, a house, a fire station, and an apartment complex. However, it is all collapsed (except for the house, which somehow survived the quake), and there might be survivors waiting to be found among the rubble! Authorities decide to send their robot first responder to generate a map of the area so that they may asses the damage caused and look for survivors.

3.2 Robot features

The first responder robot sports two wheels, two spherical casters, an RGB-D camera, a hokuyo laser sensor, and a small enough body to fit into narrow spaces. One limitation might be that this robot was not designed to be all-terrain - a feature that can certainly come in handy for rescue bots. A later, more advanced, deployment-ready version should definitely consider this as a necessary addition to the robot model.

3.3 Package Structure

To facilitate the access to the files for launching within ROS, the user will find all the .launch files under the ”launch” folder, all the .world files under the ”worlds” folder, and the single script used (teleop) under the ”scripts” folder.

To launch the Gazebo world source the `setup.bash` and run the following command from the `catkin_ws` directory.

```
$ roslaunch slam_project
world.launch
```

This command will spawn the robot in the `earthquake_rescue.world` world. To change the world file, open the `world.launch` file under the ”worlds” directory and change the `world_file` argument when defining arguments for the node (at the top of the document).

To launch the rviz configuration, run the following command on another terminal:

```
$ roslaunch slam_project
rviz.launch
```

This command will use the configuration saved under the `slam_project/launch/config` directory.

To teleoperate the robot, open up another terminal and run

```
$ roslaunch slam_project
teleop.launch
```

This launch file uses the teleop script to let the user teleoperate the robot with the keyboard.

Finally, to run RTAB-Map, run

```
$ roslaunch slam_project
mapping.launch
```

in another terminal. Now mapping has begun and it is only necessary to navigate around

the environment to generate the map! It will be automatically saved under the `.ros` directory in the user's home folder.

Additionally, the maps and databases created from both the kitchen dining world and the earthquake rescue world can be found under the `slam_project/maps/` directory.

4 Results

So at first it was hard to get everything connected properly to get the robot to actually start mapping. A link for the RGBD camera frame - along with its respective joint - had to be added and it was necessary to make sure all topics were properly connected. However, once this was achieved, the rest of the project was easier to complete. It was particularly fun coming up with a meaningful application in which such a tool could be used and then build a world to illustrate such a situation.

NOTE: both maps shown here were generated with three passes of the robot on similar trajectories throughout the environments.

4.1 Kitchen Dining World

The first environment consisted on a small part of a house with wood tile floors, a walkway through what might be the front door of the house, a kitchen/dining table, and a small living room. So this is probably the bottom floor of a house or apartment. Be that as it may, notice that this world is feature-rich - fancy word for "has many edges we can use in the bag-of-words to make loop closures!" Therefore, the maps that result from this world are quite detailed (see appendix figures 4 and 3).

As it has been explained, loop closures are critical to accurate mapping. Therefore the number of loop closures shown in the database are very important to determine whether the created (or current) world is good enough for mapping or not. Another measure that might help determine whether the world is feature-rich is the number of yellow spots on the photos taken by the robot. Many yellow spots means many features!

Thus, the results from the mapping of the kitchen dining world are rich in number loop closures (69: as seen in figure 3) and the result of the 3D map (see figure 4) represents the given world quite accurately.

4.2 Earthquake Rescue World

An interesting challenge with this project was the rescue concept. If this robot is to look for people in the rubble after an earthquake or other emergency, then obviously the map must be detailed enough to recognize people that might be lying around the place that need assistance. Also, as a second priority, the map should be good enough to asses the damage caused by the disaster. These challenges are an interesting test for RTAB-Map.

After finishing mapping, the robot was able to make a map in which people are clearly recognizable (see figures 7 and 8)! This is impressive because one of the two people that were placed in the environment was not at the same height as the robot, but a bit higher on a little hill of rubble. Despite this challenge, the point cloud seen in Rviz while mapping was precise enough for any human controlling the rescue robot to realize that a person was there. Of course, once it has found a person who needs help, the current version of the robot can't do anything about it, but this will be discussed in more detail in a future section.

Again, it is time to determine the quality of the mapping for this environment by looking at the database generated by RTAB-Map. Take a look at the number of loop closures - 468: higher even than those of the last world (see figure 9). Also, the 3D map generated (figure 10) is consistent (although definitely and understandably imperfect) compared to the world (figures 6 and 5).

5 Discussion

Now, it is time to assess what worked and what didn't work. Obviously, nothing is perfect, and there is room for improvement in these tasks.

5.1 Kitchen Dining World

5.1.1 What didn't work

In this case, despite the 69 loop closures obtained by the RTAB-Map algorithm, it would

seem that the map is rather inexact in some things - such as exact positions of some chairs, the fridge, and the table in the living room (figure 4). However, it must be said that if the map's objective is to generate an occupancy grid (shown in figure 3), then the results of the package are definitely good enough to work with, even if the detail of the map isn't precise down to every inch.

5.1.2 What worked

What did work in this map was to have the robot follow almost the same trajectory three times. That gave the algorithm enough confidence in the data to generate a detailed map that resembles the provided world in a measure good enough to avoid collisions in a future navigation and path planning task.

5.2 Earthquake Rescue World

5.2.1 What didn't work

The objective of this robot was twofold: both being able to find people in the environment who might need help and being able to assess the damage caused by a natural disaster. As it turns out, the degree of achievement of the second objective seems to lag behind that of the first. The damage proves difficult to asses from the data obtained by the robot's generated map. Sure, a few piles of rubble are definitely visible in the 3D map, but it is not clear if the rubble belongs to the apartment complex, the fire station, or both.

The reason for this flaw seems to be the robot's height - it is too short to see if the buildings were destroyed and how bad the damage is! It can only see the rubble. Perhaps an improvement that could be made in order to achieve a more comprehensive mapping of the world that could help determine the damage could be a drone instead of a wheeled robot, although any robot that is tall enough to see the buildings might work.

5.2.2 What worked

What definitely worked was the inclusion of humans in the 3D map of the world! They are clearly distinguishable in figures 7 and 8. Their location with respect to the map could be determined easily with this data, helping first responders to quickly and efficiently rescue these people if needed.

Another good result worth noting is the occupancy grid! Granted, it might be missing a few things (such as better representing the tree and the dumpster), but these limitations could be easily fixed simply by paying more attention to them while producing the map - both the tree and the dumpster were largely disregarded while performing the mapping task, unlike the box, which might look funny in the 3D map (figure 10), but is clearly not listed as navigable in the occupancy grid (figure 9).

This means that this robot, although potentially incapable of correctly providing information for damage assessment after a natural disaster, can pave the road quite well for other robots who might be able to do it - since they could now navigate more easily, only performing localization with the given occupancy grid while scanning specifically for damage. Not only that, but it can also determine how much of a priority this location is for first responders - since the number of humans in the world can be determined accurately. Thus, the robot seems to live up to the name "first responder" in that it can do a lot of heavy lifting so that humans don't have to - instead of wasting rescuers in areas where there might not be any people who need help, humans (as well as more specialized rescue bots) can focus only on priority areas which this little bot can determine without risking lives.

It is also worth noting that, although this particular world was built with a more open space to facilitate visualization of the robot's movements in the simulator, more narrow places leave less space for error, and the size of this robot small exactly to allow it to navigate where humans might not be able to. Thus the limitation of the map's accuracy might not be such a big deal in real deployment, although further testing is necessary to determine this for sure.

6 Future Work

Further improvements of this robot (other than actual deployment on hardware) include the

possibility of adding rescuing capabilities so that it can perform the rescue directly - how to go about that is a great question that requires thought and further research and innovation.

Perhaps adding a movable camera joint to examine other angles of the environment (and thus assess the damage) could be an interesting addition, although the way the angles will affect the point cloud generated by the point cloud of the robot might lead to complications while a larger robot might be unable to fit in more narrow, inaccessible places despite the fact that it may be able to assess damage better.

Other areas where mapping might be useful include planetary exploration, home service, and scouting for the army. All of these approaches are interesting - but saving lives seems like a much cooler approach!

APPENDIX

The following pages contain images of the package's resulting graphics and data.

It is recommended to view these results using the `rtabmap-databaseViewer` command. This provides better tools for visualization, as well as the possibility to generate the 3D map while running.

To open the database generated by RTAB-Map for the kitchen dining world, run the following command from the `catkin_ws` directory:

```
$ rtabmap-databaseViewer src/slam_project/maps/rtabamap.db
```

To view the 3D map generated by the database, go to "Edit" and then click on "View 3D map..."

It might take a while, but the 3D map will be displayed.

Unfortunately, the project was too heavy to also upload the earthquake rescue database and point cloud, but the pictures included aim to provide an idea of how it looks.

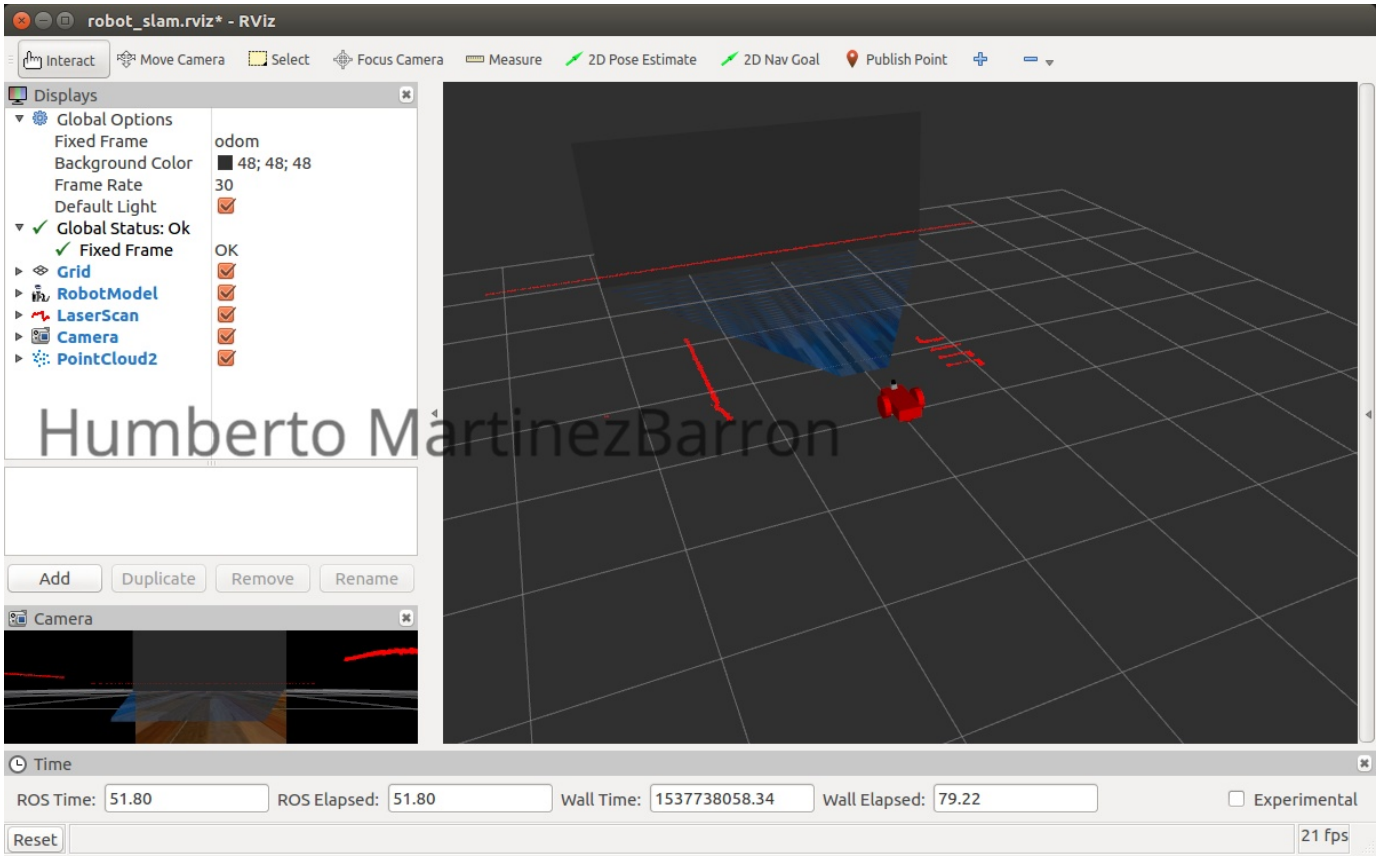

Figure 1. The robot's boot as seen in Rviz for the kitchen dining world.

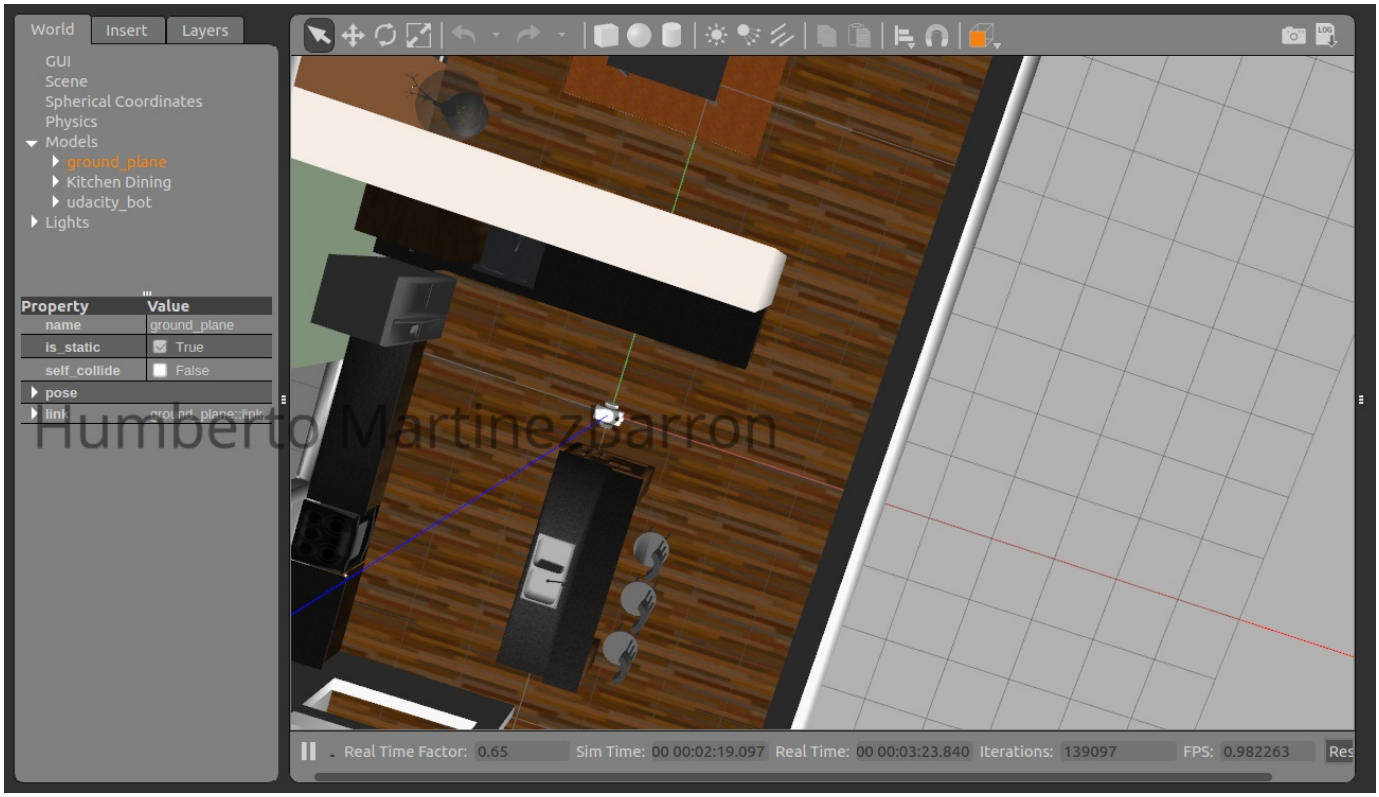

Figure 2. Booting up the robot on gazebo looks like this for the kitchen dining world.

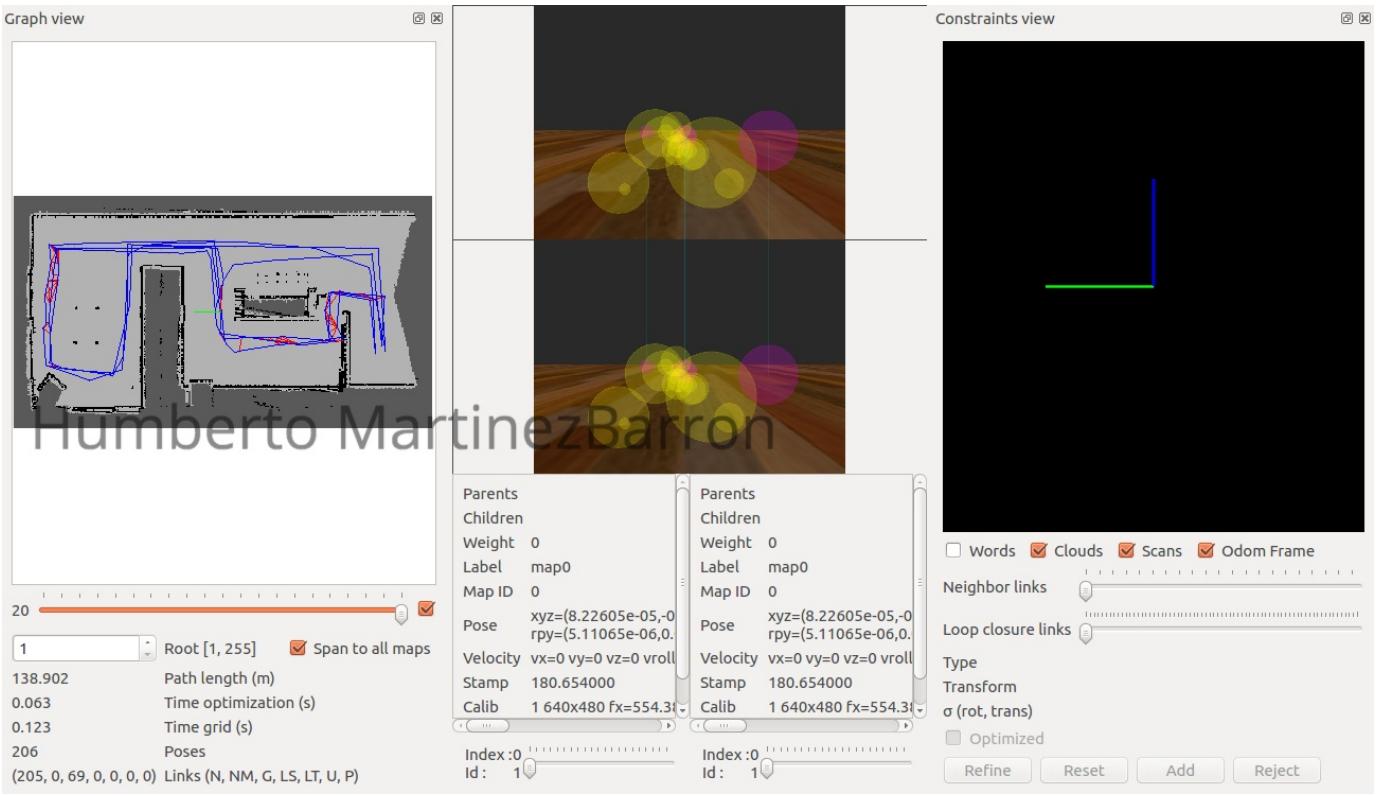

Figure 3. 2D map, constraints, and sample image from the database generated for the kitchen dining world.

Figure 4. The cloud generated by RTAB-Map for the kitchen dining world.

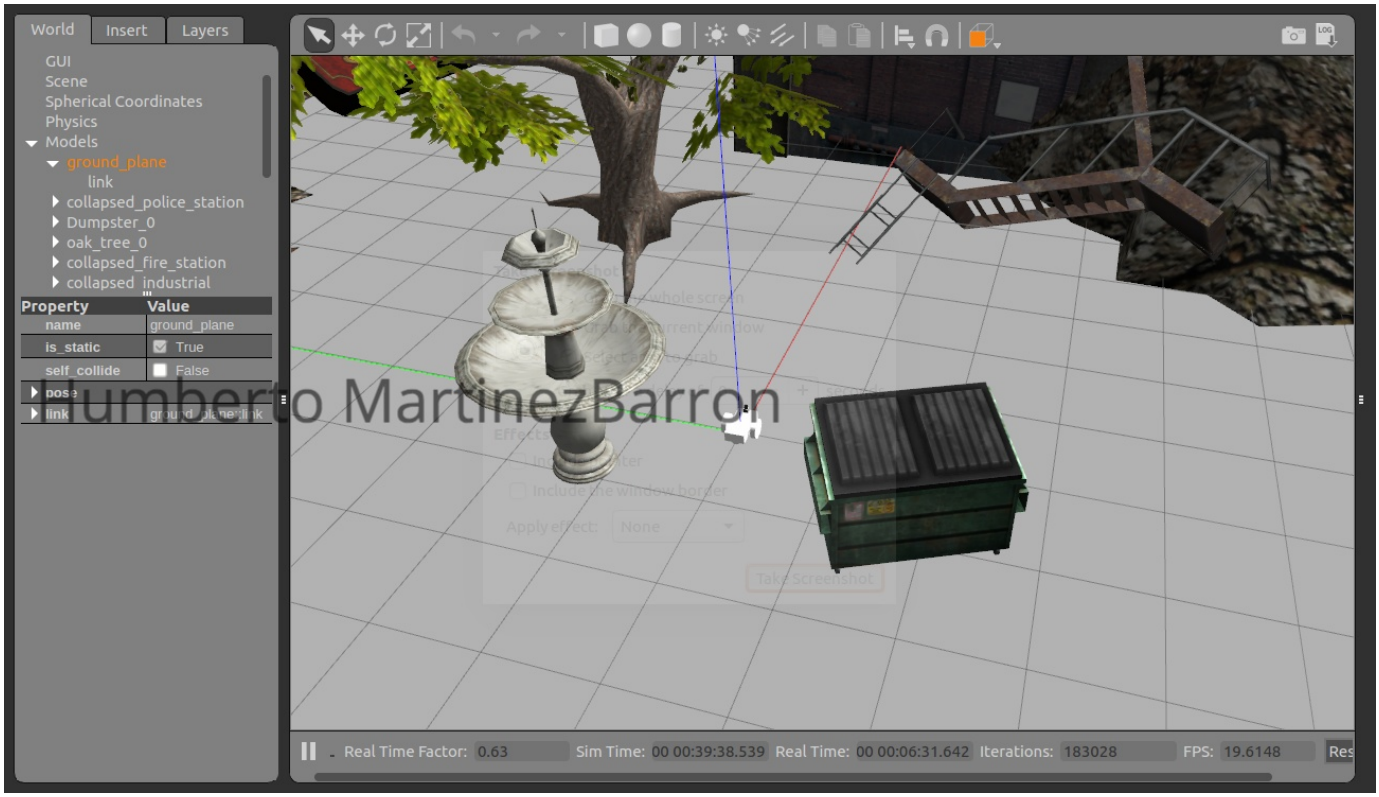

Figure 5. The spawning of the robot in gazebo for the earthquake rescue world.

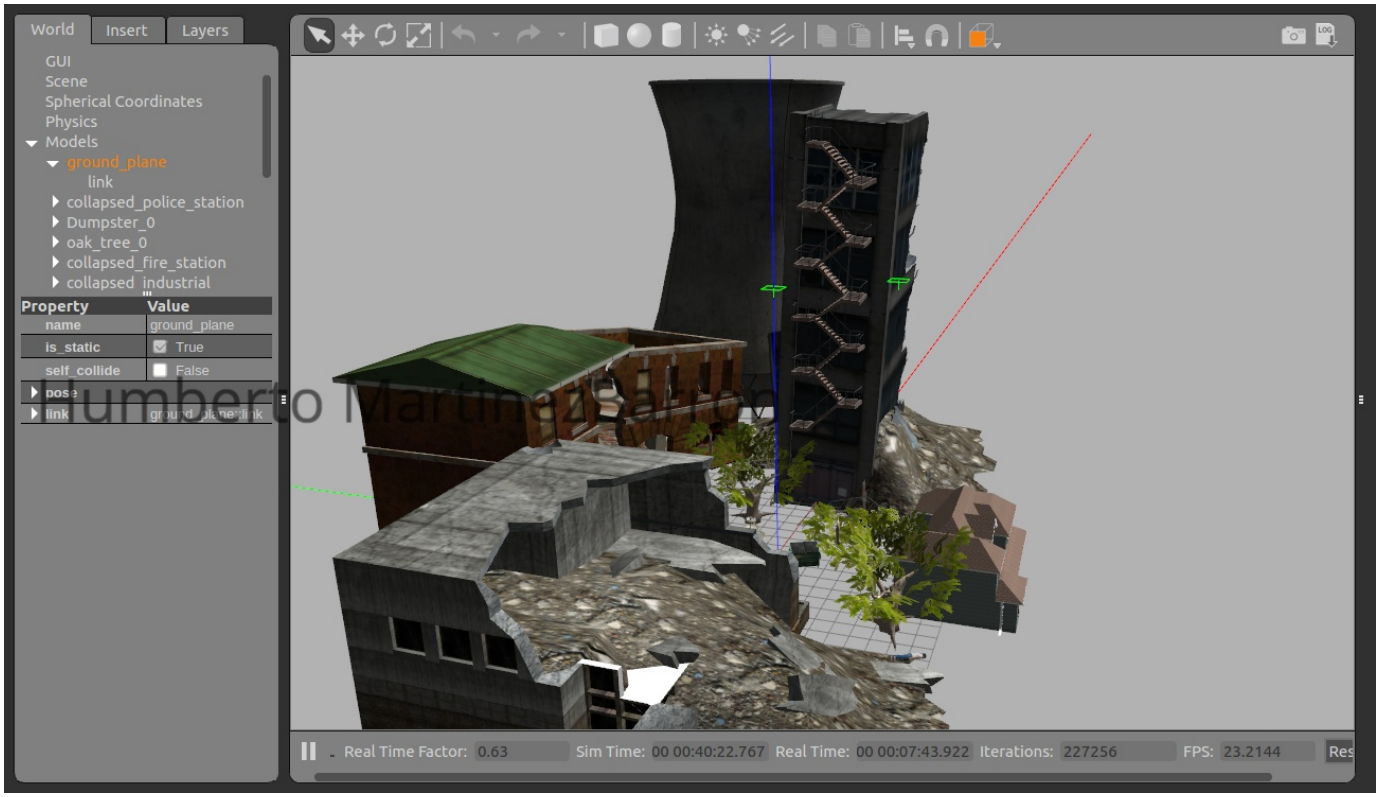

Figure 6. The complete world seen from a different angle to provide a view of what is included in the world.

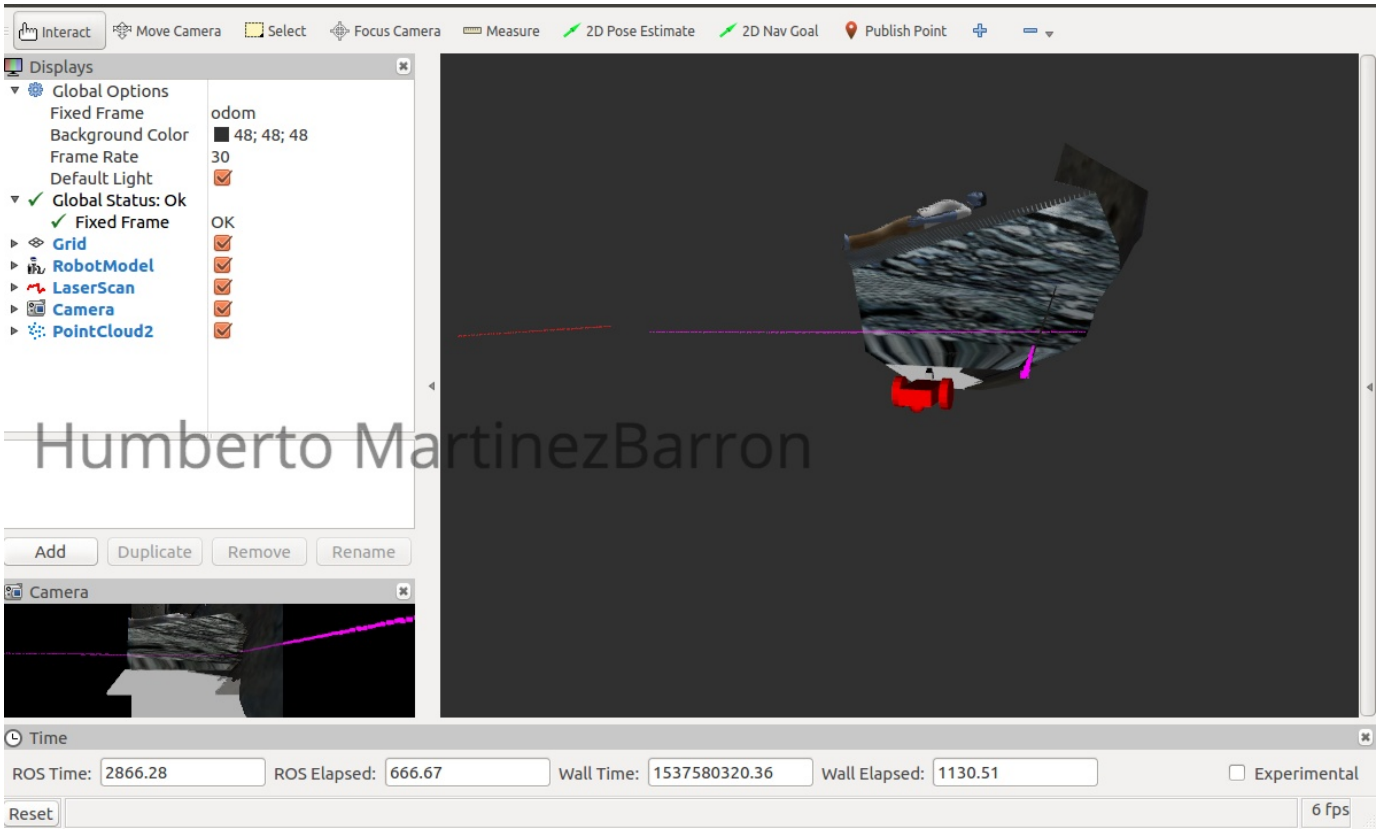

Figure 7. The robot seen in Rviz in the rescue world finding a person.

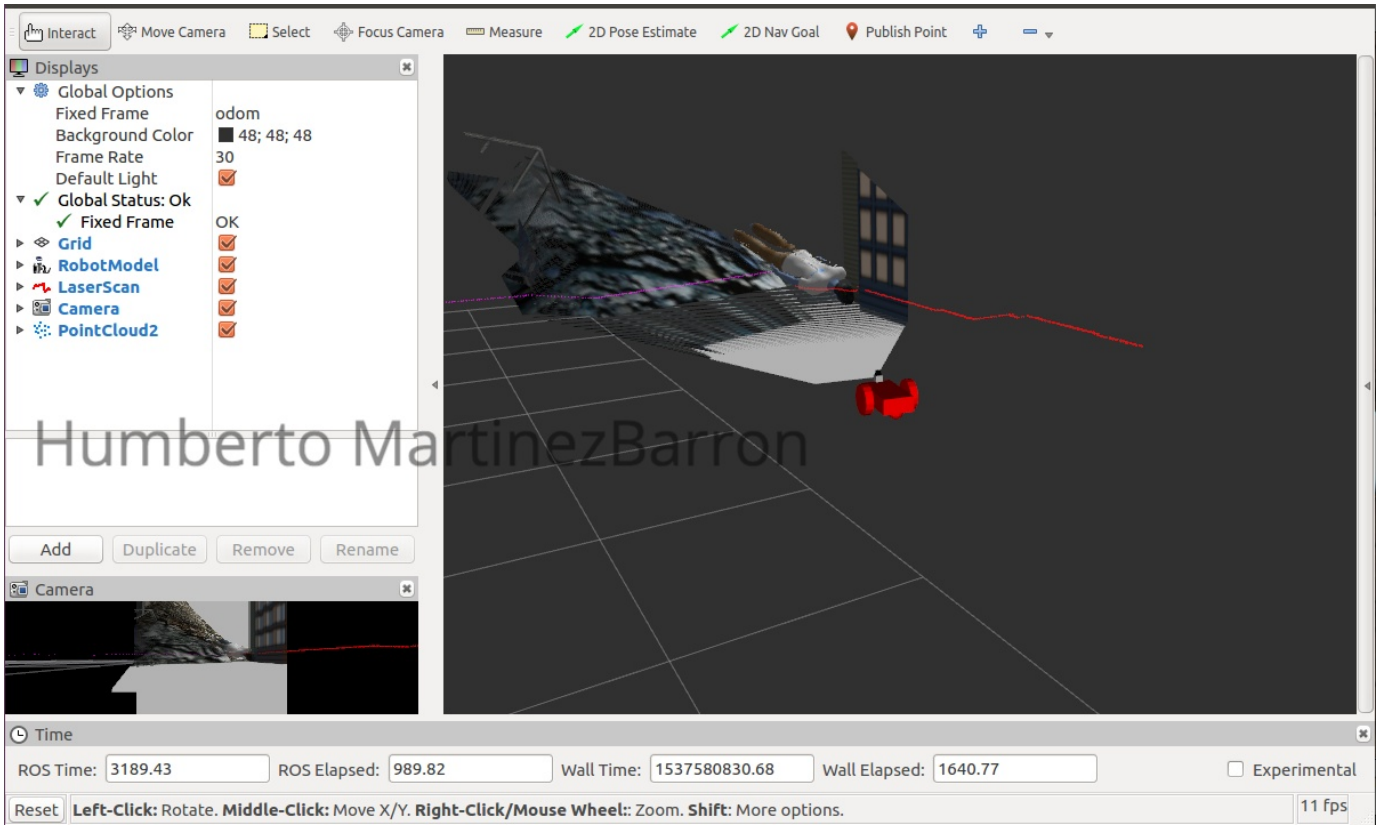

Figure 8. The robot seen in Rviz finding the second person.

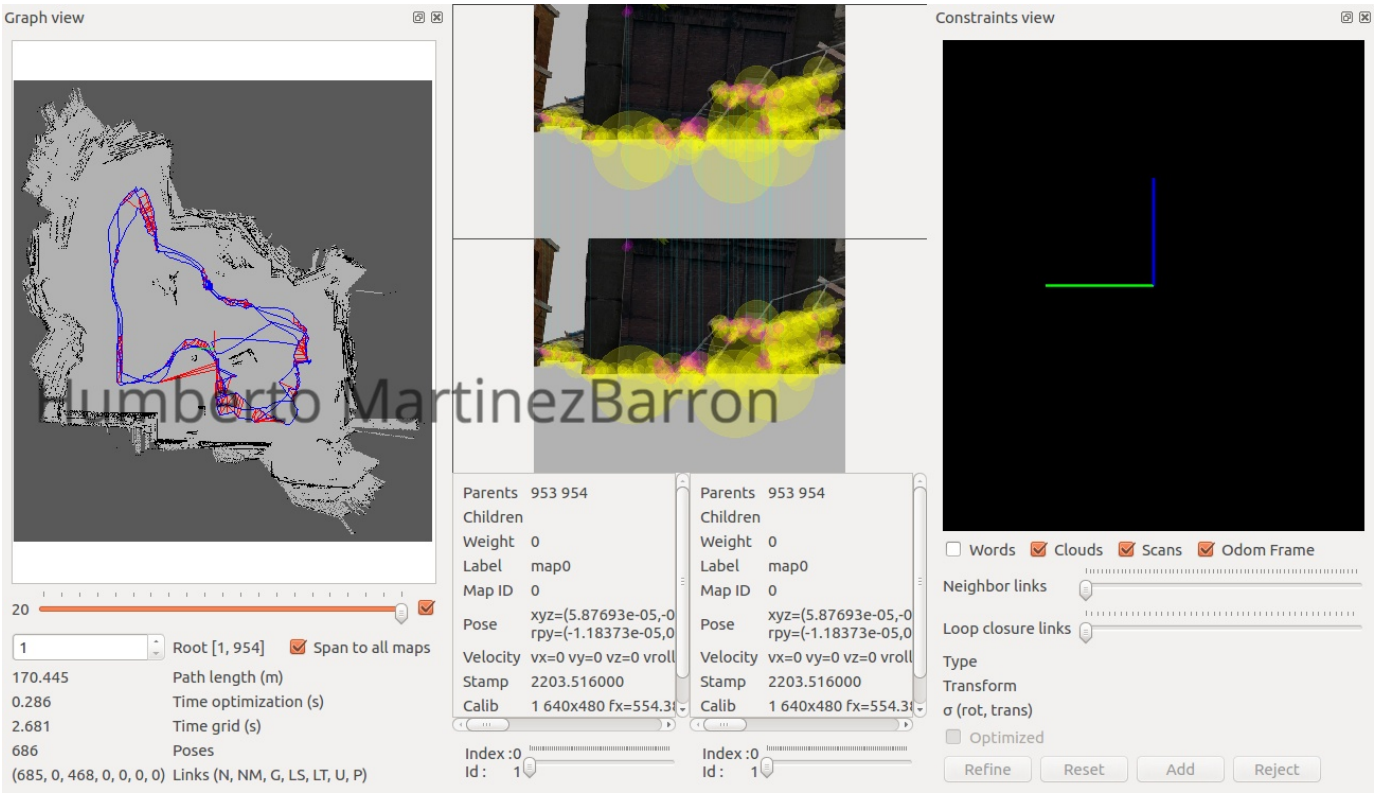

Figure 9. The 2D map, constraints, and sample images generated for the rescue world.

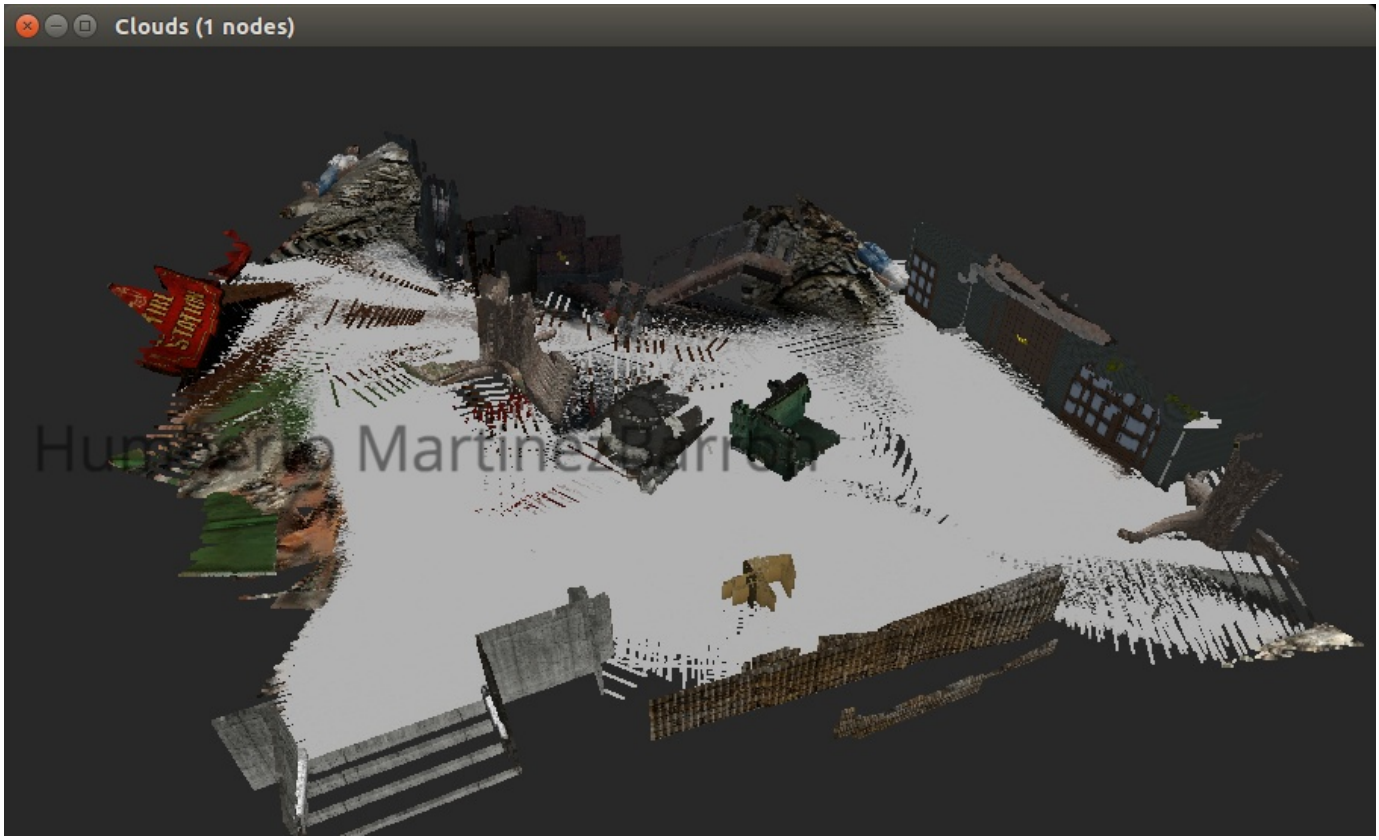

Figure 10. The point cloud generated by RTAB-Map in the earthquake rescue world.

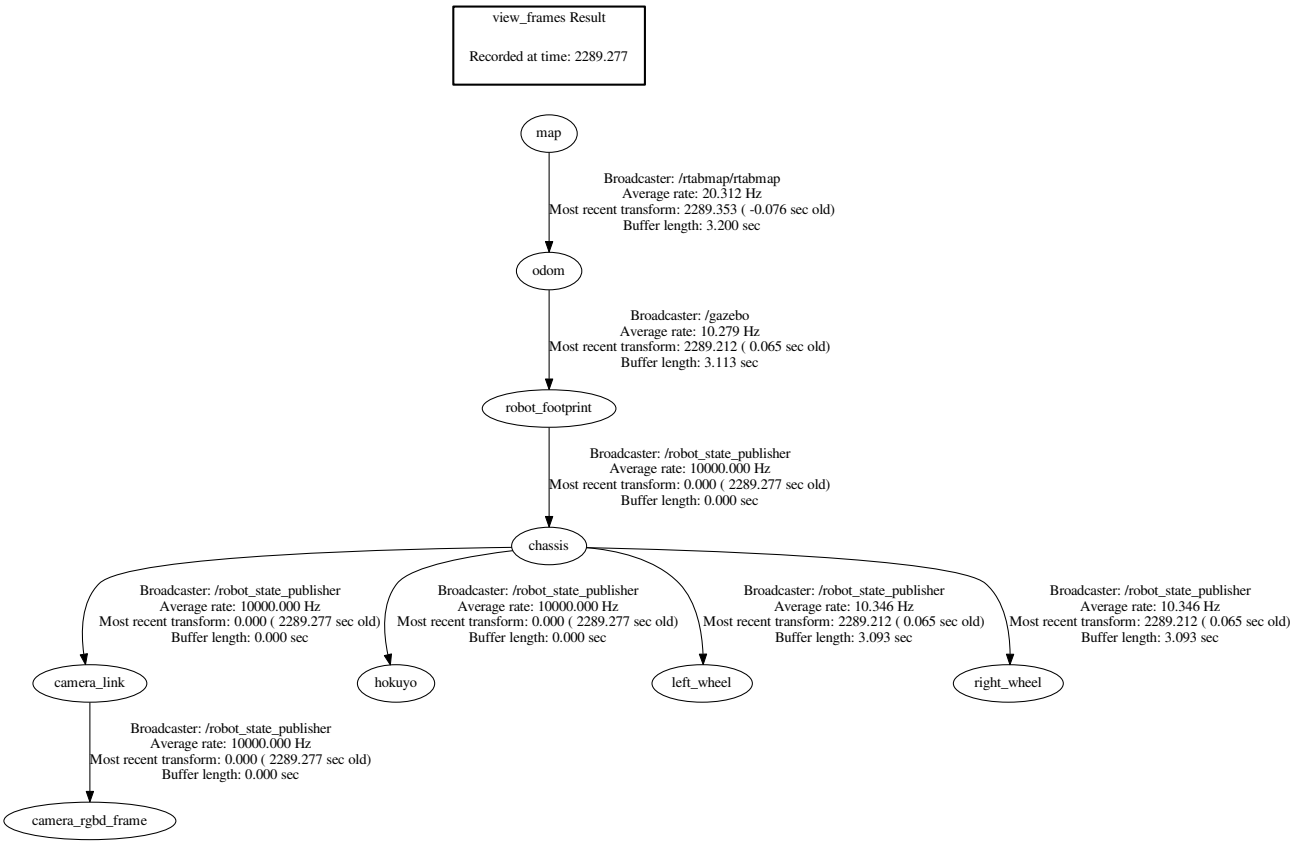

Figure 11. The transformed frames tree of the robot's link connections.