Universiteit Antwerpen

Progress Report Week 13

Michiel Aernouts
michiel.aernouts@student.uantwerpen.be

May 20, 2017

Abstract

Optimisation of wireless networks is critical for the localisation of wireless devices. For this purpose, a wave propagation model of the environment can be created. Such a model contains a map of the environment combined with RF measurements that are obtained within that map. In this paper, we compare several visual SLAM algorithms such as LSD SLAM [1] and RGB-D SLAM [2] that can be used to render an accurate 3D map of an indoor environment. In order to test these algorithms, simulation software is used to navigate a drone around a room. A camera that is mounted on the drone provides necessary data for the algorithms. After finishing a SLAM algorithm, the resulting point cloud can be implemented in an OctoMap [3] to generate a volumetric representation. An initial guess of the environment is modelled and merged with the OctoMap that resulted from the SLAM algorithm. This way, an accurate probabilistic model of the environment can be created.

1 Progress

1.1 Merging OctoMaps

This week, I successfully created a probabilistic map that contains the initial guess model as well as SLAM measurements. The `octomap_merger` launch file contains a number of steps to obtain this result. Figure 1 shows a basic schematic that describes my approach.

1.1.1 Initial Guess

I had to find a method to show my initial guess model as an Octomap in RViz. My initial idea was to write an octomap publisher that reads the `.bt` file an generates a ROS message that could be viewed in RViz as an occupancy grid. However, this approach came with a few drawbacks. The `octomap-server` resolution and the `.bt` file resolution would have to

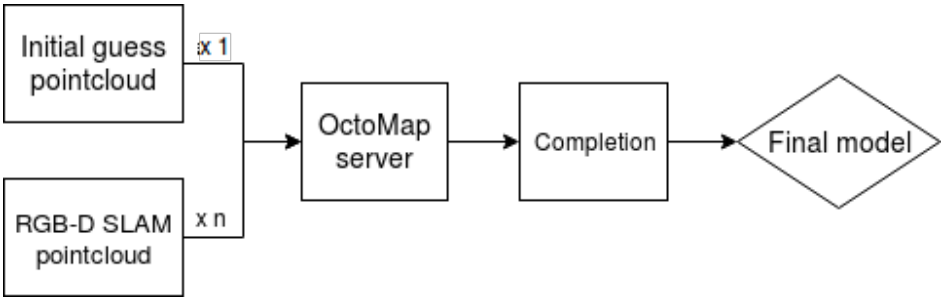


Figure 1: Basic schematic for my map merging approach.

be set to the exact same values. Also, it would be difficult to update the initial guess model with the RGB-SLAM point cloud, because they are not published on the same ROS topics. Therefore, I now implement a different approach. First, I convert the `.bt` file to a `.pcd` point cloud by executing the `bt2pcd` code. The resulting `.pcd` file is then published to the `/cloud_pcd` topic with the `pcd_to_pointcloud` ROS node. It is important to remap the `/cloud_pcd` topic to `/cloud_in`, as `octomap-server` listens to the latter topic. The map can be viewed on the `/initial_map` frame in RViz.

1.1.2 RGB-D SLAM

In order to update the initial guess model with real measurements, we use the RGB-D SLAM point clouds, which are published on the `/rgbdslam/online_clouds` topic. For use with OctoMap, this topic has to be remapped to `/cloud_in`. Furthermore, it is important to correctly set the TF information settings of RGB-D SLAM. First, `/fixed_frame_name` should be set to `/map` so that the point cloud can be seen on this frame. Second, `/base_frame_name` should be set to `/base_link`. This will tell RGB-D SLAM the transformation of the robot base.

1.1.3 Map Alignment

Both point clouds have to be transformed so that they are perfectly aligned. Only then will we able to achieve an accurate merged map. For this purpose, I transformed the `/initial_map` frame to the `/map` frame by empirically comparing the Octomaps and then changing the rotation and transformation of the initial guess model. I use the `static_transform_publisher` ROS node for this. Take note that the process of map alignment will be more accurate if a high-resolution Octomap is rendered.

1.1.4 Octomap Server

As mentioned before, `octomap-server` will listen to the `/cloud_in` topic and render a probabilistic model. First, the initial guess point cloud is published once. Second, the RGB-D SLAM algorithm publishes a point cloud to the `/cloud_in` topic. As both point clouds are processed through the same topic, they will be merged into a single probabilistic map.

1.1.5 Completion

For now, no further steps are taken to complete the Octomap. I will experiment with:

- Publishing the initial guess model multiple times to give it more weight in the occupancy grid.
- Filling holes with nearest neighbour exploration algorithms.

1.1.6 First results

The first result of my approach is displayed in figure 2. Figure 2a exhibits the initial guess model before updates from RGB-D SLAM. Figure 2b shows the result after updating. Only the rooms were recorded in the dataset, so it is obvious that the other 2 rooms were not updated.
For now, I can only draw empirical conclusions about this map. Knowing the environment that was recorded, figure 2b appears to be an accurate representation. At the window-side, voxels appear to be detected outside the initial guess model. This is probably due to a slight error in the modelling of the initial guess, I will measure and update the dimensions before testing again.
Furthermore, I will have to test and quantify which parameters have an influence on accuracy. Some parameters that come to mind are:

- Octomap resolution
- Number of times the initial guess cloud is published
- RGB-D SLAM
  - Min and max point cloud depth that is processed
  - Feature extractor/detector type (ORB or SIFTGPU)
  - Max amount of extracted keypoints
  - Point cloud density
  - Backend solver used in g2o

1.2 Gazebo

In order to test the accuracy of a SLAM algorithm, we can use Gazebo. The initial guess model can be recorded in simulation. Afterwards, the SLAM algorithm has to process the recordings. The resulting point cloud or octree can be compared to the actual model. By doing this, we will have a better understanding of how much noise the SLAM algorithm possibly adds to the environment, and we can conclude how accurate the algorithm is.

1.2.1 Import model in Gazebo

Our model can be imported as a mesh in Gazebo by following these steps:

- Convert the `.stl` file to a Collada file (`.dae`). I used an online converter for this: `http://www.greentoken.de/onlineconv/`.
- Put the Collada file in the same folder as your `.world` file for Gazebo.
- Include the model in your `.world` file. Example in figure 3.

1.2.2 Yaw rotation

In previous simulations, I was not able to rotate the drone in a stable fashion. This was a main disadvantage when testing visual SLAM algorithms.
This week, I finally solved this problem. Apparently, the `rotors_simulator` ROS package that is used by Erle Robotics is out of date with the new versions. This package provides multimotor models for simulating a multicopter. The solution to the yaw rotation problem was eventually found at `https://github.com/ethz-asl/rotors_simulator/commit/71c1ddef1ff42345090a2dd8e112683ecf7ac46f`. I applied the changes in my local `rotors_simulator` package and built it again. After that, the simulated Erle-Copter had no problems rotating! Next week, I can use the improved simulator to run proper tests for RGB-D SLAM and LSD SLAM.

1.2.3 Trajectory scripts

As yaw rotation is now working, trajectories can be scripted for our simulated model. I wrote some C++ code that executes a basic trajectory around the inital guess model. However, this is still a work in progress with a low priority.

2 Planning week 14

- Start to put the final paper together

  - Formulate a better title

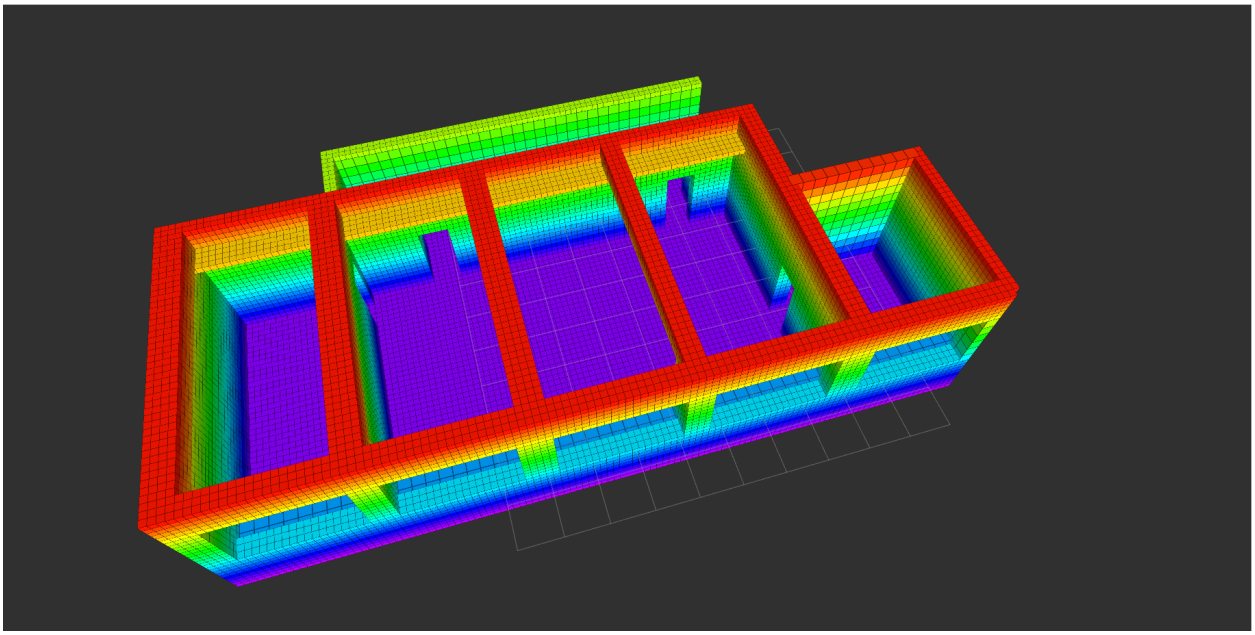
- RGB-D SLAM accuracy test in Gazebo
- Map completion
- LSD SLAM test with a wide fov camera
- Quantify accuracy results

References

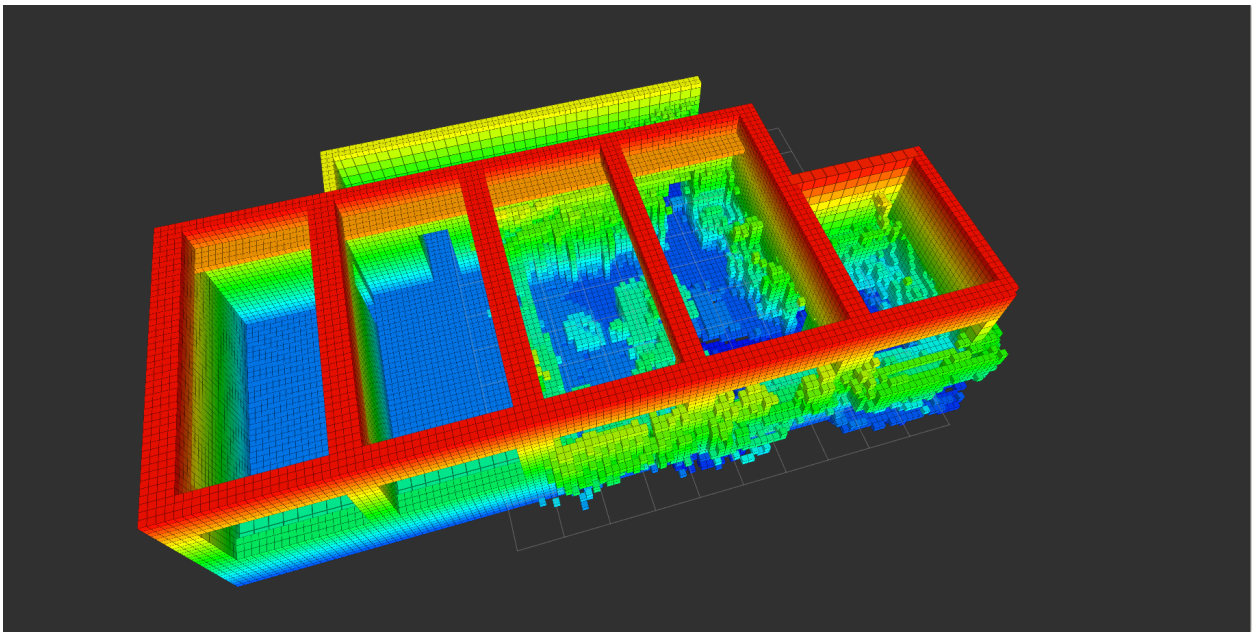
[1] Jakob Engel, Thomas Schöps, and Daniel Cremers. LSD-SLAM: Large-Scale Direct Monocular SLAM. *Computer Vision ECCV 2014*, pages 834–849, 2014.

[2] Felix Endres, Jurgen Hess, Nikolas Engelhard, Jurgen Sturm, Daniel Cremers, and Wolfram Burgard. An evaluation of the {RGB}-D {SLAM} system. *2012 {IEEE} International Conference on Robotics and Automation*, 2012.

[3] Armin Hornung, Kai M Wurm, Maren Bennewitz, Cyrill Stachniss, and Wolfram Burgard. {OctoMap}: an efficient probabilistic {3D} mapping framework based on octrees. *Autonomous Robots*, 34(3):189–206, 2 2013.



(a) Before updating the initial guess with real measurements



(b) After updating the initial guess with real measurements

Figure 2: Result of the approach, with an Octomap resolution of 15 cm

```
<model name="v_building">
  <pose>-6 2 -0.2 0 0 -1.5708</pose>
  <static>true</static>
  <link name="body">
    <visual name="visual">
      <geometry>
        <mesh><uri>model://vbuilding/complete_no_ceiling.dae</uri></mesh>
      </geometry>
    </visual>
  </link>
</model>
```

Figure 3: Example to include a model into a `.world` file