

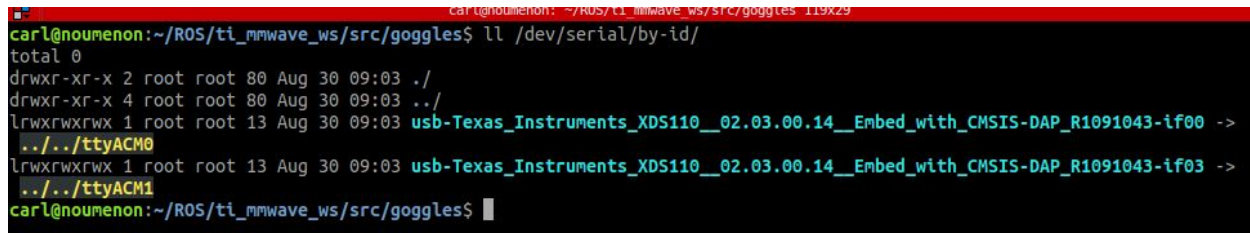
Radar Octomap Setup Guide

Carl Stahoviak

carl.stahoviak@colorado.edu

NOTE: These instructions assume that the TI AWR1843BOOST is recognized as a USB device. This can be verified by running the following command in the terminal:

```
ll /dev/serial/by-id
```



```
carl@noumenon: ~/ROS/ti_mmwave_ws/src/goggles 119x29
carl@noumenon:~/ROS/ti_mmwave_ws/src/goggles$ ll /dev/serial/by-id/
total 0
drwxr-xr-x 2 root root 80 Aug 30 09:03 ./
drwxr-xr-x 4 root root 80 Aug 30 09:03 ../
lrwxrwxrwx 1 root root 13 Aug 30 09:03 usb-Texas_Instruments_XDS110_02.03.00.14__Embed_with_CMSIS-DAP_R1091043-if00 ->
.././ttyACM0
lrwxrwxrwx 1 root root 13 Aug 30 09:03 usb-Texas_Instruments_XDS110_02.03.00.14__Embed_with_CMSIS-DAP_R1091043-if03 ->
.././ttyACM1
carl@noumenon:~/ROS/ti_mmwave_ws/src/goggles$
```

These instructions also assume that you have installed the following ROS packages. If the board is not properly recognized, or if the following packages have not been installed, please refer to the ARPG Goggles Setup Guide, which can be found [here](#).

1. [ti_mmwave_ropkg](#) (ARPG version)
2. [goggles](#)
3. [serial](#)

This instruction guide takes you through the following process:

1. Cloning the `radar_rig` and Andrew's forked and updated `octomap_mapping` ROS packages
2. A brief description of `octomap.launch` and the parameters available for tuning
3. Launching the Octomap server node

1. Cloning the `radar_rig` and Associated Packages

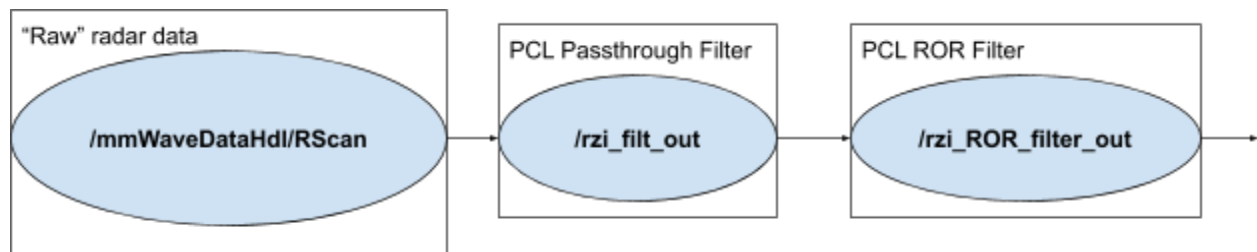
In the `src` folder of a catkin workspace where `ti_mmwave_ropkg`, `serial`, and `goggles` are located, clone the following repos:

```
git clone https://github.com/cstahoviak/radar_rig.git
git clone https://github.com/1988kramer/octomap_mapping.git
```

2. Octomap.launch basics & Parameters

The radar octomap utility takes the following 3-step approach to generate an Octomap from radar data.

1. The “raw” radar pointcloud2 message (`/mmWaveDataHdl/RScan`) is first filtered using a [PCL Passthrough filter](#) to remove targets outside of specified range, elevation and intensity ranges. This is done by `radar_limit_filters.launch`.
2. The output of the PCL PASsthrough Filter (`/rzi_filt_out`) is filtered again using the [PCL Radius Outlier Removal \(ROR\) filter](#) and is output on the `/rzi_ROR_filter_out` topic.
3. The Octomap server is launched with `/rzi_ROR_filter_out` as the `cloud_in` topic, and the occupied cells (`/radar/occupied_cells_vis_array`) are visualized using RVIZ.



where the final arrow is what gets piped to the Octomap Server as the `cloud_in` topic

Parameter Tuning:

1. **PCL Passthrough Filter:** the range, elevation, and intensity ranges can all be tuned
2. **PCL ROR Filter:** the `min_neighbors` and `radius_search` parameters (as described in more detail in the link above) have already been set to appropriate values for radar data, but these can be tuned to alter the performance of the ROR filter.

3. Launching the Radar Octomap Server:

There are two methods for using `octomap.launch` to start an Octomap Serder:

1. **‘Online’ mode:** online mode will launch the TI radar device, the Goggles velocity estimator, and any number of additional sensors (camera, lidar, imu, etc) as specified in lines 10-15 of `octomap.launch`

```
roslaunch radar_rig octomap.launch
```

2. **'Offline'** mode: assumes that the `/mmWaveDataHdl/RScan` topic (and optionally the MLESAC inliers topic `/mmWaveDataHdl/inliers` - an output of the Goggles node) is already started (either as the result of re-playing data from a bag file, or that the radar has already been launched from a separate launch file).

```
roslaunch radar_rig octomap.launch offline:=true
```

NOTE (on odometry): As with any use of Octomap, you will be responsible for providing a static transform from the radar sensor frame (`base_radar_link`) to an estimated odometry/pose frame. And then a non-static transform from the pose/odom frame to the map frame (See lines 73-77 of `octomap.launch`). Currently, `octomap.launch` relies on the pose estimate from the Intel T265 Tracking Camera, but this can be replaced with a package such as `robot_localization` that will do pose estimation from the velocity output (and optionally IMU data) from the Goggles node. This is not provided as part of the `radar_rig` package (yet).

An example Radar-Octomap (with T265 odometry) result is shown below. Drift is evident by the misalignment of the part of the hallway in the lower-left corner of the map.

