Camera Drones Lecture 2023-24
Jesús Pestana <jesus.pestana@pro2future.at>
Friedrich Fraundorfer <fraundorfer@icg.tugraz.at>

Practical Part 1

ROS related theory

- ROS architecture
- ROS master, nodes, and topics
- Console commands
- Catkin workspace and build system
- Launch-files
- RViz visualization
- ROS bag
- rqt – ROS bag inspection
- tf-tree – coordinate transformations between rigid-body frames

Task Description

The goal of this task is to generate an *obstacle map* for the drone from a *ROS bag* provided by us (see the data section) – a ROS bag is a ROS specific datalog. The provided ROS bag mainly contains RGB-D images (RGB images + Depth) and their corresponding camera poses acquired using a motion capture system at the "dronespace" of the TU Graz. Note: *obstacle maps* have the main focus of representing the presence of occupied space in 3D coordinates around the robot and permit the planning of collision-free trajectories (also called obstacle-free trajectories), that is, trajectories in the free space around the robot. Obstacle maps can be specified in *3D space* or in *Configuration Space* (C-Space) [R9].

Please, read all the document before proceeding with the proposed steps for the task. In particular, the "tips" section provides details / support on how to perform these steps. Note that ROS, due to its code being divided in separate "packages", lacks a comprehensive documentation. The "tips" section is intended to ease learning on the usage of ROS. If you find yourself stuck for a "long" time, please do not hesitate to contact the tutor of the ROS practicals via E-mail or by using the teach-center forum.

The creation of the map (or obstacle map) will be done primarily using the *octomap_server* node [R1][V1][R2] (results shown in Fig. 1). However, some alternatives are provided below – in the ROS Packages section – for those of you that want to experiment further on this topic.

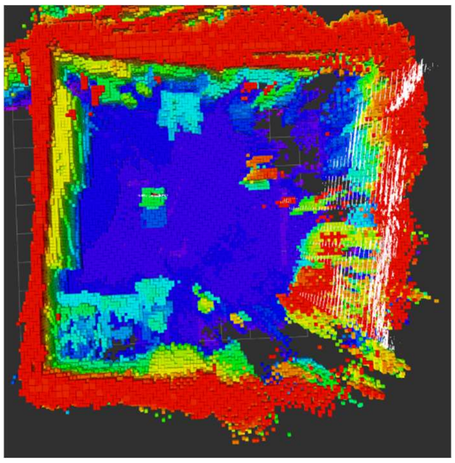


Fig. 1 Octomap obtained from processing the ROS bag 2019-11-18-19-37-10_bag

Jesús Pestana <jesus.pestana@pro2future.at>
Friedrich Fraundorfer <fraundorfer@icg.tugraz.at>

In order to get started with the task, first prepare a catkin workspace with the ros_tutorials package [R8], compile this workspace and source it automatically in the "${HOME}/.bashrc" shell script. This script is executed automatically each time you open a terminal window, instructions on its basic configuration can be found in [R5]. Alternatively: use the catkin workspace from the ROS tutorial lecture (contained in the virtual machine), which is already for you to start working on the practicals.

The following steps are advised to achieve the map / obstacle-map creation task:

1. Inspect the data in the ROS bag using *rqt* [R6]
2. Visualize the data in *ROS RViz* [R7] (2D images and *tf* position data of the drone)
3. Read the documentation about the octomap_server node and set up the octomap_server *ROS launchfile* to configure the node and make it connect to the right topics from the ROS bag. Note: ROS bags can be played back using the *rosbag play* executable.
4. Create an obstacle map using the octomap_server node and the provided ROS bag, set the resolution of the octomap to between 10-30 cm
5. Visualize the octomap in ROS RViz
6. Store the created octomap as a binary (e.g. bt file) for use in the next tasks.

Alternative packages that can be used to generate an obstacle map are listed in the "ROS Packages" section of this document.

Tips

The following are some tips about steps at which you will probably need further information:

- First, it is a good idea to check which is the input of the octomap_server node, see: http://wiki.ros.org/octomap_server
  The input to this node is the point cloud (cloud_in) from the RGB-D sensor plus its corresponding positions specified as tf transforms. The relevant frames / reference systems are the pose of the map frame (name specified on the octomap_server launchfile) and the – moving – frame of the depth sensor (specified on the header of the cloud_in topic messages – message type sensor_msgs/PointCloud2).
- The point-clouds of the sensor can be visualized in Rviz (http://wiki.ros.org/rviz) once the tf transforms are available, see Fig. 3. For this you need to select in Rviz the appropriate global frame (usually "world" or "map") and the tf-tree needs to have enough information to provide the position of the depth sensor/camera with respect to the selected global frame.
- Several tf transforms are specified in the tf_static topic. Static transforms do not need to be re-published frequently, which makes them efficient, but they are problematic when playing back ROS bags / datalogs (they are only published once at the very beginning of the ROS bag playback). Inspect the ROS bag using rqt to confirm this issue. To avoid issues, when playing back data logs (rosbags), it is better to rerun the static transform publishers:
    - For the transforms of the parts of our RGB-D sensor (and ORBBEC Astra Pro), download its ROS Driver (and compile it):
      git clone -b 0.3.0 https://github.com/orbbec/ros_astra_camera astra_camera
      This '0.3.0' is the last tagged commit that compiles properly and for which the following roslaunch command works. Run the node as follows:
      roslaunch astra_camera astra.launch load_driver:=false publish_tf:=true –wait
      Note: This node additionally takes care of the conversion of depth images to point-clouds. Depth-maps are kept in image format until they are used, because point-

clouds comparatively require more memory and are more costly to transfer between nodes – programmes / processes – with the ROS messaging system and to store in ROS bags. Remark: exchanging ROS messages of big size is, however, not an issue when exchanging information between ROS Nodelets (in this case pointers to the data are exchanged between processes / ROS Nodelets).

- For the intersensory calibration, optitrack pose (frame: drone_1/base_link) to RGB-D sensor base link (frame: camera_link), start a static transform publisher: http://wiki.ros.org/tf2_ros#static_transform_publisher
  specifying the following transformation:
  - Translation (XYZ – m): [0.050 -0.007 -0.0375]
  - Rotation (YPR – rad): [0. -0.135625 0.0175]
- Inspect the tf-tree (rosrun tf view_frames; evince frames.pdf), it should look as follows (see Fig. 2) - or alternatively use “rqt” to visualize the tf-tree:

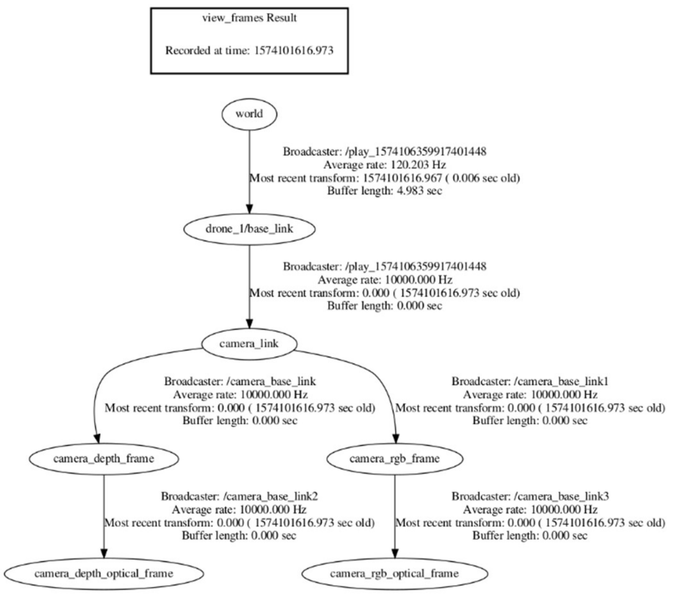
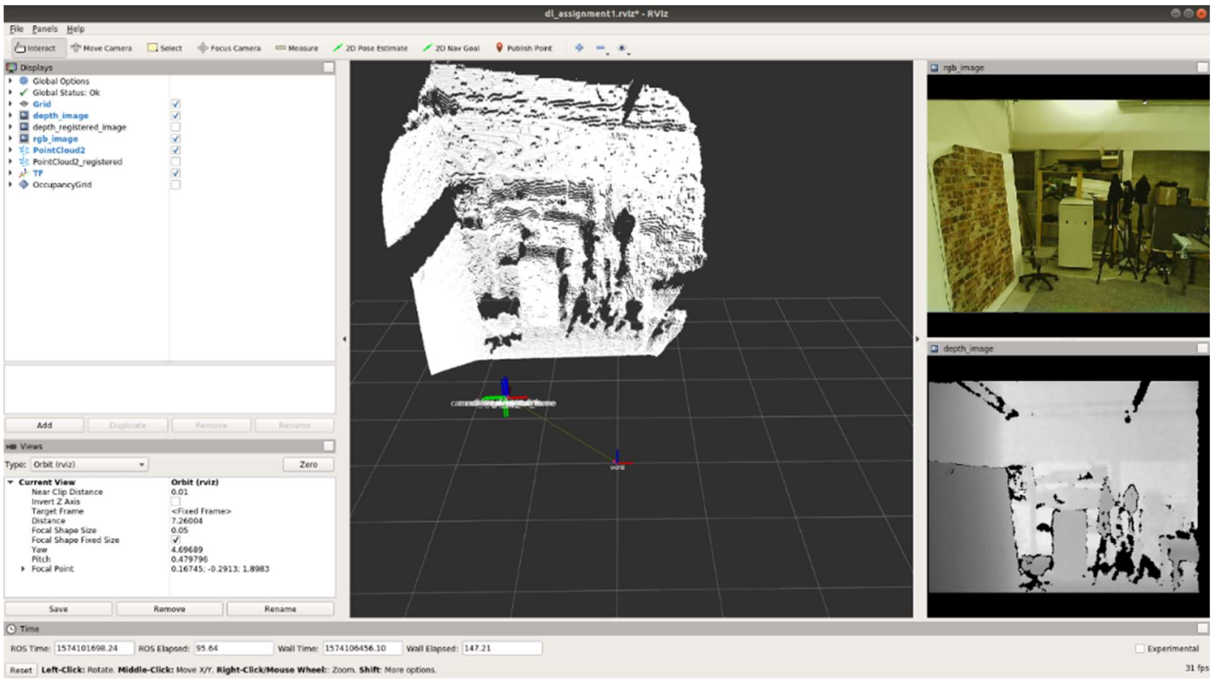

view_frames Result
Recorded at time: 1574101616.973
world
Broadcaster: /play_1574106359917401448
Average rate: 120.203 Hz
Most recent transform: 1574101616.967 (0.006 sec old)
Buffer length: 4.983 sec
drone_1/base_link
Broadcaster: /play_1574106359917401448
Average rate: 10000.000 Hz
Most recent transform: 0.000 (1574101616.973 sec old)
Buffer length: 0.000 sec
camera_link
Broadcaster: /camera_base_link
Average rate: 10000.000 Hz
Most recent transform: 0.000 (1574101616.973 sec old)
Buffer length: 0.000 sec
Broadcaster: /camera_base_link1
Average rate: 10000.000 Hz
Most recent transform: 0.000 (1574101616.973 sec old)
Buffer length: 0.000 sec
camera_depth_frame
camera_rgb_frame
Broadcaster: /camera_base_link2
Average rate: 10000.000 Hz
Most recent transform: 0.000 (1574101616.973 sec old)
Buffer length: 0.000 sec
Broadcaster: /camera_base_link3
Average rate: 10000.000 Hz
Most recent transform: 0.000 (1574101616.973 sec old)
Buffer length: 0.000 sec
camera_depth_optical_frame
camera_rgb_optical_frame

Fig. 2 tf-tree of the acquisition setup

- Visualize the point cloud data in ROS Rviz:
  - Start a roscore
  - Start ROS Rviz (rviz for short)
  - Playback the ROS bag (rosbag for short)
  - Configure rviz to visualize the data in the rosbag, in particular, the tf transforms and the point-cloud data (see the utilized “Displays” on Fig. 3).
  - In rviz, use the "world" (or "/world") frame from as fixed frame (set in the upper left corner, under “Global Options”)
  - In rviz, the tf library is buffering the frames received through the "/tf" and "/tf_static" topics. If you re-play the rosbag without restarting the tf buffer of rviz, it will complain about the timestamps not being consistent (always increasing over time). You may receive errors such as: "Lookup would require extrapolation into the past. [...]". To reset the tf buffer, there is a "reset" button on the lower left corner.

- In order for all nodes to use consistent timestamps, with those stored in your rosbag datalogs (which correspond to the past), you want all your nodes to use the timestamps recorded inside the rosbag. This is as if the current time was really the same as in the recorded messages. In order to achieve these you have to do the following 2 things:
    - Immediately after you initialize the roscore, you want to set the "use_sim_time" ROS parameter to true (before you start any other process/node, such as Rviz). To do this run the command: "rosparam set /use_sim_time true" .
    - All the nodes will now be in stand-by waiting to receive messages and timing information. You will provide the timing information from the "rosbag play" command/process, for instance with the command: "rosbag play --clock --rate=0.25 bagfile.bag". The --clock option tells the rosbag to publish the simulation time, and the --rate option tells the rosbag to playback the messages (and the corresponding time topic) at a slower or faster rate than the "real-time" recording.



Type: Orbit (rviz)

Current View	Orbit (rviz)
Near Clip Distance	0.01
Invert Z Axis	☐
Target Frame	<Fixed Frame>
Distance	7.26004
Focal Shape Size	0.05
Focal Shape Fixed Size	✓
Yaw	4.69689
Pitch	0.479796
Focal Point	0.16745; -0.2913; 1.8983

ROS Time: 1574101698.24 ROS Elapsed: 95.64 Wall Time: 1574106456.10 Wall Elapsed: 147.21

Left-Click: Rotate. **Middle-Click:** Move X/Y. **Right-Click/Mouse Wheel:**: Zoom. **Shift**: More options.

Fig. 3 Visualization of tf-frames, images and point cloud data in Rviz. On the left side of the image, the data displays that I have used are listed. For the occupancy grid I use the OcuppancyGrid data display type from the package ros-melodic-octomap-rviz-plugins.

- Download the octomap_mapping repository (and compile it), and configure the octomap_server node properly by modifying the launchfile octomap_server/launch/octomap_mapping.launch . Note: alternatively, just download the launchfile and install the octomap_mapping package from the Ubuntu repositories. The name of the package in the ubuntu repositories is probably “ros-*distro*-octomap-mapping” (distro is either melodic or noetic). Launchfiles specify the the nodes to be started (and their corresponding pacakages). Therefore, launchfiles can be stored elsewhere (not necessarily on the launch folder of a specific package).

ROS Packages

- (recommended) octomap [R1], octomap_mapping & dynamicEDT3D [R2]
- InfiniTAM [R3]
- Voxblox [R4]

Note-1: both {octomap + dynamicEDT3D} and Voxblox provide an obstacle map with distance-to-obstacle pre-calculated data for points in the free space of the obstacle map. These data are necessary later on to plan obstacle-free trajectories for the drone using less computing time / resources.

Note-2: InfiniTAM is mainly a fast 3D reconstruction software, which is more oriented towards creating visually appealing and accurate 3D models efficiently. Voxblox is, at least, partially motivated on InfiniTAM. In contrast the Octomap library was designed to reflect the presence of obstacles in 3D space using voxel Octrees [R10] (which looks a bit like environments in Minecraft).

Data

A rosbag from a recording of the droneSpace has been prepared to be used with your algorithms. You can download it using the following direct link:

2019-11-18-19-37-10.bag: https://files.icg.tugraz.at/f/fb3643f14540402895d5/

Grading Scheme, assignment 1 score: 15 out of 60 points

The submission needs to include a small report – PowerPoint or PDF slides – about your work on this assignment. No source-code needs to be submitted for this assignment.

Most of the score obtained for this assignment (75%) is decided from the following content:

- Image(s) showing your obtained Octomap (similar to Fig. 1 above).
- (Optional) Video (screen recording) or images showing the incremental Octomap generation.

Choose any 2 points to work on from the following, which will decide the rest of your score (25%):

- Image of perfect tf-tree (similar to Fig. 2 above).
- Rosbag inspection (e.g., using rqt-bag or rosbag info in the terminal).
- Benchmarking of the size of the obtained Octomap binary file, ".bt", for instance, with respect to the resolution of the Octomap (minimum octant/cube size).
- You can optionally discuss in detail issues you have encountered, or any other work which you have done for the assignment.

References

[R1] Hornung, A., Wurm, K. M., Bennewitz, M., Stachniss, C., & Burgard, W. (2013). OctoMap: An efficient probabilistic 3D mapping framework based on octrees. *Autonomous robots*, *34*(3), 189-206. Link: http://wiki.ros.org/octomap_server

[V1] [ROS Q&A] 085 - Basic usage of octomap_mapping package - Tutorial https://youtu.be/dF2mlKJqkUg

[R2] Lau, B., Sprunk, C., & Burgard, W. (2013). Efficient grid-based spatial representations for robot

navigation in dynamic environments. *Robotics and Autonomous Systems*, *61*(10), 1116-1130.
Link: https://docs.ros.org/melodic/api/dynamic_edt_3d/html/

[R3a] Kähler, O., Prisacariu, V. A., Ren, C. Y., Sun, X., Torr, P., & Murray, D. (2015). Very high frame rate volumetric integration of depth images on mobile devices. *IEEE transactions on visualization and computer graphics*, *21*(11), 1241-1250.
Link: https://github.com/ethz-asl/infinitam

[R3b] Kähler, O., Prisacariu, V. A., & Murray, D. W. (2016, October). Real-time large-scale dense 3D reconstruction with loop closure. In *European Conference on Computer Vision* (pp. 500-516). Springer, Cham.
Link: https://github.com/ethz-asl/infinitam

[R4] Oleynikova, H., Taylor, Z., Fehr, M., Siegwart, R., & Nieto, J. (2017, September). Voxblox: Incremental 3d euclidean signed distance fields for on-board mav planning. In *2017 Ieee/rsj International Conference on Intelligent Robots and Systems (iros)* (pp. 1366-1373). IEEE.
Link: https://github.com/ethz-asl/voxblox
Link: https://github.com/ethz-asl/mav_voxblox_planning

[R5] Ubuntu install of ROS Melodic - 1.5 Environment setup
Link:
http://wiki.ros.org/melodic/Installation/Ubuntu#melodic.2FInstallation.2FDebEnvironment.Environment_setup

[R6] ROS rqt - A Qt-based framework for GUI development for ROS
Link: http://wiki.ros.org/rqt

[R7] ROS Rviz - 3D visualization tool for ROS
Link: http://wiki.ros.org/rviz

[R8] ROS Tutorials Package
Link: https://github.com/ros/ros_tutorials

[R9] Wikipedia – Motion Planning – Configuration Space
Link: https://en.wikipedia.org/wiki/Motion_planning#Configuration_space

[R10] Wikipedia – Octree
Link: https://en.wikipedia.org/wiki/Octree