ME 132A - LAB 3 - QUADROTOR PLANNING

Released: Wednesday 02/22/2017 Due: Friday 03/10/2017

This guide describes how to design a simple planner for a quadcopter. It starts by getting familiar with the virtual environment we are going to use and the set of libraries ROS. The learning curve for ROS may be very steep at the beginning but we encourage you to do the tutorial in their webpage[1]. The guide will start by using the simulator environment Gazebo, embedded in ROS. This tool will allow us to test code without the risk of crashing the vehicle. It will show how to do a open planner, a simple controller and a point to point planner. Once the code in the simulator is working, the students will allocate a time slot with the TAs to test their code in hardware.

Install Hector quadrotor

1. Install Hector Quadrotor using the *install_lab3.sh*. It will:
   (a) Install Hector Quadrotor, following the wiki http://wiki.ros.org/hector_quadrotor/Tutorials/Quadrotor%20indoor%20SLAM%20demo
   (b) Install a keyboard controller.
2. If you are running this on your own machine instead of the virtual machine you may encounter these errors:
   (a) `Error [Node.cc:90] No namespace found` Make sure Gazebo is included in the system path. There is an error when it is trying to get the models from the repository. To solve this issue download all models by tiping:

```
$ wget -r -R "index\.html*" http://models.gazebosim.org/
$ sudo sh -c 'echo "deb http://packages.osrfoundation.org/gazebo/ubuntu
    trusty main" > /etc/apt/sources.list.d/gazebo-latest.list'
```

   Check that you execute the simulation in the same folder you have downloaded the files.
   (b) `ImportError: No module named rospkg`: if you used python in the computer the path might be corrupted for ROS. Check that the python path is not corrupted by typing:

```
$ export PYTHONPATH=$PYTHONPATH:/usr/lib/python2.7/dist-packages
```

   Another known errors are related with internet connectivity issues and outdated Linux distributions. Make sure you have connection and the Linux distribution is updated (run $ sudo apt-get update)
3. Analyze the default configuration of the simulation. This will be very useful later to debug your code.
   (a) Open the outdoor demo (it may take a while).

[1] http://wiki.ros.org/ROS/Tutorials

```
$ roslaunch hector_quadrotor_demo outdoor_flight_gazebo.launch
```

You should see two windows, one for Gazebo and another for RVIZ. Check the errors in the terminal. To stop a ROS node just press Ctrl+C.

(b) Use *ros_graph* to analyze the topic/node structure by typing the following command in a new terminal. Use this tool everytime you change the code to debut it visually. Make sure the nodes are connected as expected

```
$ rqt_graph
```

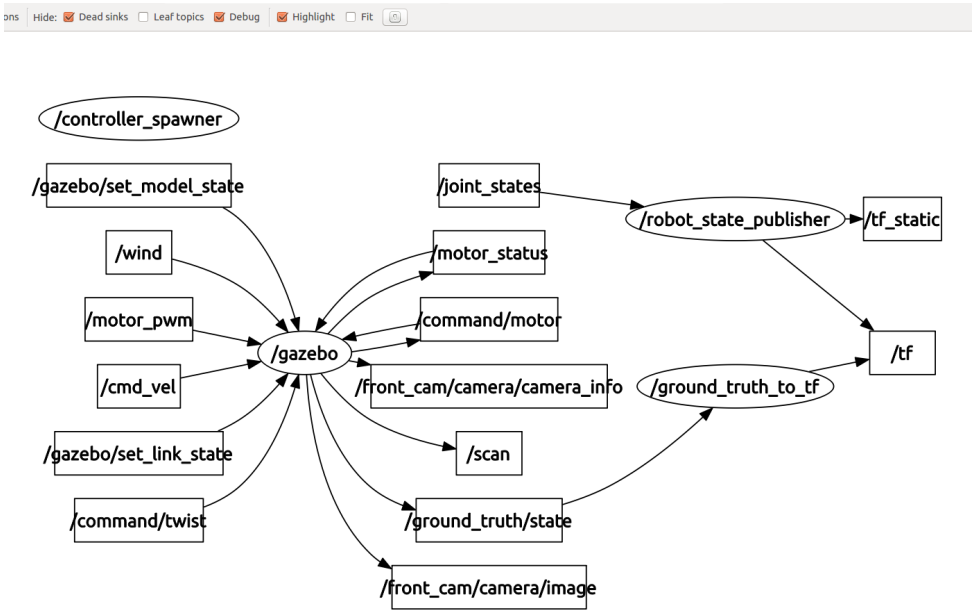

Figure 1: Rqt_graph for the original simulation.

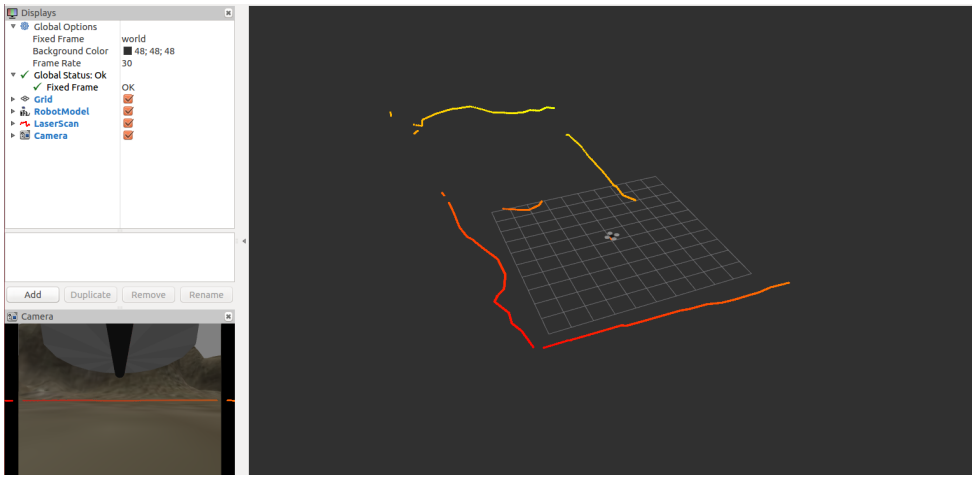

Figure 2: Screen shoot of outdoor_flight_gazebo.

(c) Later we will close the diagram by reading /tf and publishing /cmd_vel. To see all tranformations in /tf, open *rqt* by just typing $ rqt in a new terminal. Open Plugins -> Visualization -> TF Tree. It will show something similar to Figure 3c.

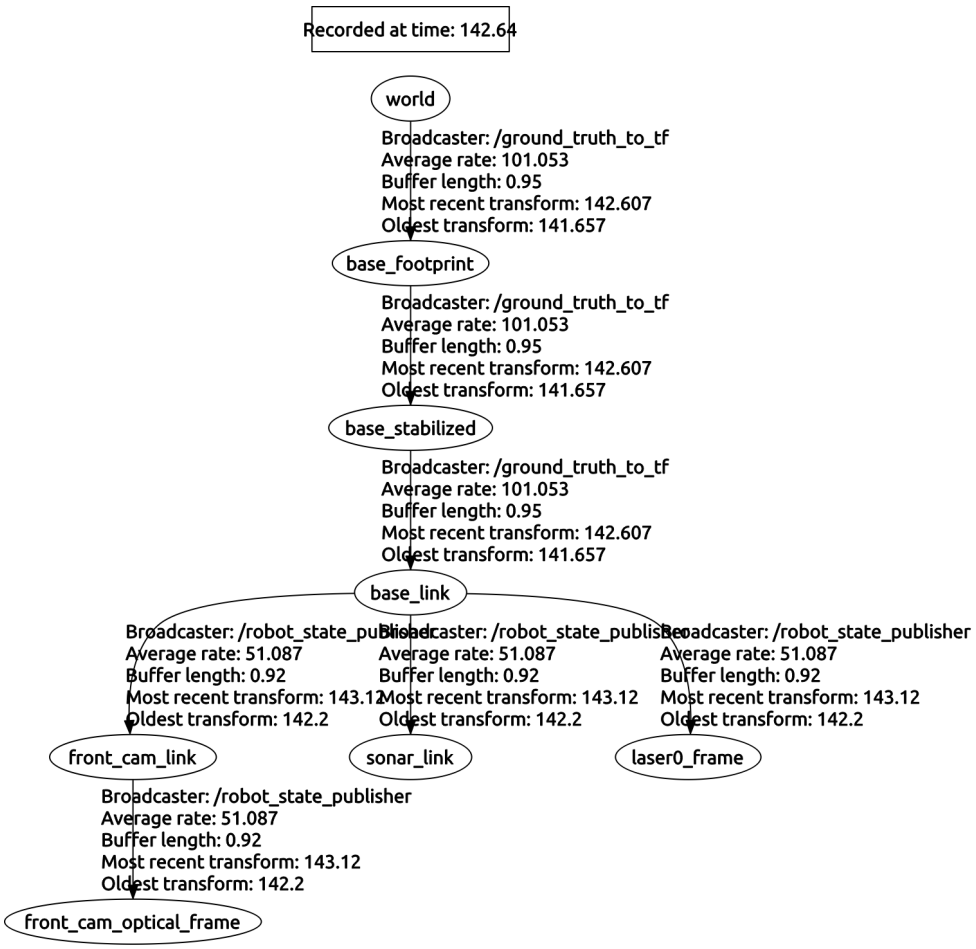

Figure 3: Transformation frames hierarchy.

4. Instantiate a keyboard controller and fly the drone around. This will send commands to cmd_vel The keys are written on the terminal.Type in a new terminal:

```
$ rosrun teleop_twist_keyboard teleop_twist_keyboard.py
```

Note: before using any other commands you must first take off using the button 't'. Check your new *rqt_grah*,it should look like Figure 4. See the new node at the left.

5. If you want to manually fly in a more challenging environment you can take a look to the indoor environment we will use for next homework as seen below, but for this week's tasks we will be using the outdoor environment.

```
$ roslaunch hector_quadrotor_demo indoor_slam_gazebo.launch
```

Create Open Loop Planner

1. The first task is to create a node that makes the drone follow simple circle in open loop. You should publish to the /cmd_vel topic. There is a template called open_loop_template.py that might be useful. You should copy it to a catkin workspace (follow the tutorials to create a catkin workspace and a package) and compile it as described here. Don't forget to make the file executable! There are also two copies of the tutorials files listener and talker for anyone

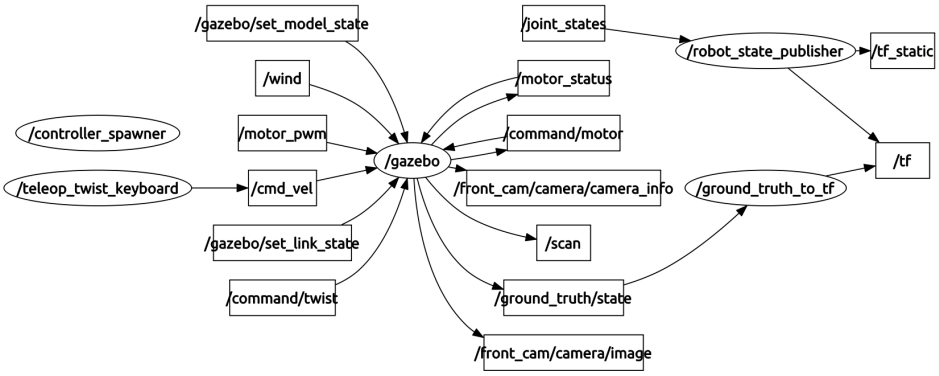

Figure 4: Rqt_graph using the keyboard and gazebo.

new to ROS. Your rqt_graph should look similar to the case with a keyboard. Run your node by

```
$ rosrun _packageName_ open_loop_template.py
```

Some guidelines to understand how to use a topic in ROS:

- Check the message type of your topic. You can use the terminal command $ rostopic type _nameTopic_. Another approach is to use rqt topic visualization. Once you know your topic type, you can analyze the message structure by the command

```
$ rosmsg show _msgName_
```

Another tool to debug your code is

```
rospy.loginfo( _variableToPrint_)
```

Create Closed Loop Planner

The next task is to read the transformation frame to correct for errors. This allows us to create a controller in the simplified dynamics of the vehicle given by the inner control running inside the drone.

- What is the relationship between this simplified dynamics and the original dynamics?
- What are the advantages and drawbacks of using this simplified dynamics

Write a node that reads the transformation frame between the world and the drone. Use the open loop template as starting point. Your new rqt_graph should look like Figure 5. Use this node to write a simple mission, 30 seconds, of the drone following that path. The student should design a simple path in 3D and then test that the drone can follow it in simulation. A simple PD should suffice to track the error between the desired path and the current position from tf. Some ideas are: your name in 3D, Lorenz Attractor, Bernoulli spiral.. The path should be contained within a cube of size S (use $S = 10m$ for initial testing).

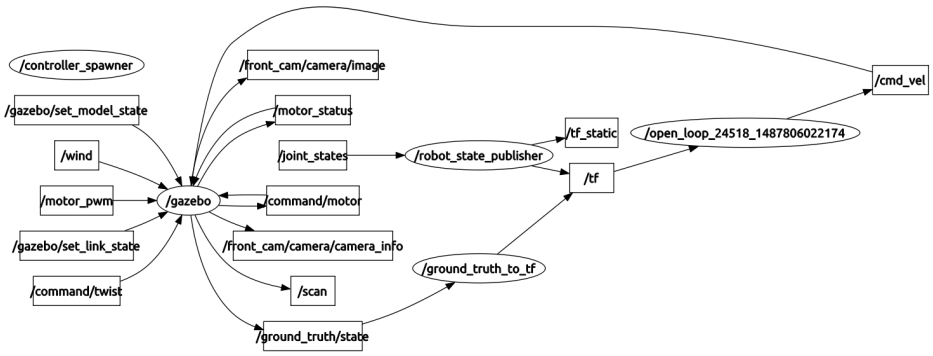

Figure 5: Rqt_graph for the closed loop system.

Hardware Testing

We are using the platform Bebop 2. It is a light quadrotor designed as ready-to-fly use. We connect to it using the published API for Parrot Drones. It allows us to retrieve odometry data, camera images and send velocity commands. If you need more information there is more in their website.

1. **Smartphone control:** the easiest way to start using the drone is by the the App, available for iPhone and Android. Connect the drone and take a demo flight. What are the inputs to the drone?

Next, you should connect the drone to your computer by running the bebop_driver

```
$ roslaunch bebop_driver bebop_node.launch
```

If you cannot connect check that you are connected to the drone WIFI. As usual with ROS, analyze the topics and the rqt_graph. What are the differences with respect to the simulator?

1. **Take-off and landing:** the first step is to take-off and land (horizontally) using the computer and ROS. Open a rqt terminal and the plugin Topics -> Message Publisher. Create a new instances for topics takeoff, land and cmd_vel. Your screen should look similar to Figure 1

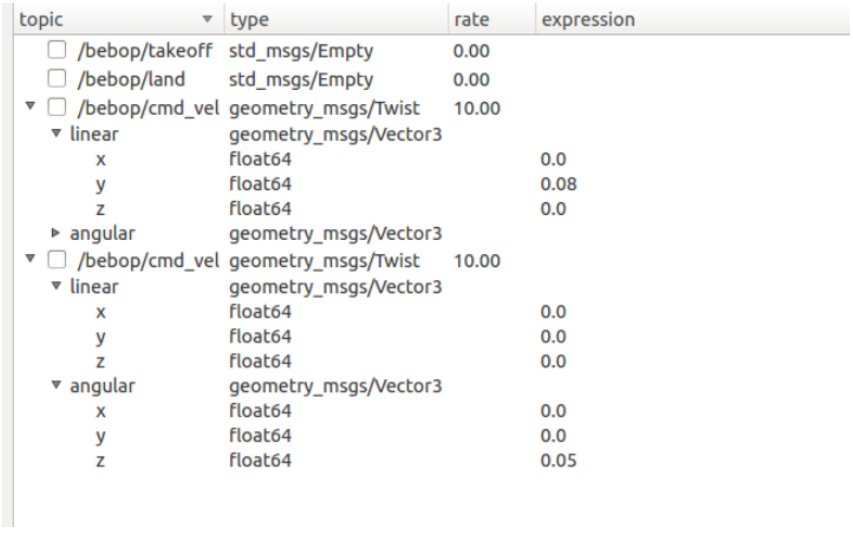

Figure 6: Predefined messages to operate the drone.

2. **Joystick control:** it allows smoother control than the keyboard. Run in a new terminal:

```
$ roslaunch bebop_tools joy_teleop.launch
```

3. **Zombie mode:** flight the quadrotor in manual flight and then play the commands back using rosbag. Note: be ready to take control using the joystick at any moment. Be ready to kill all the communication with the drone and establish a new communication (maybe using the smartphone)

   (a) Put the drone in a flat area easy to identify. We are going to take off from the same spot later.

   (b) Start the drone node

   (c) Start the joystick node

   (d) Start recording the topics using rosbag record

```
$ rosbag record -O _bagName_ /bebop/cmd_vel /bebop/land /bebop/
odometry /bebop/takeoff /bebop/image_raw
```

   (e) Flight around 1 min and then land.

   (f) Stop recording by killing the terminal with ctrl+c

   (g) Check the new bag file with rosbag info. Register the number of messages for each topic in the report.

```
$ rosbag info _bagName_
```

   (h) Come back to the original position and play the registered bag

```
$ rosbag play _bagName_
```

   How far is the landing spot of the replay commands compared to the original landing spot? Why that difference?

Homework Deliverables

1. Show the results on your work on the original simulator. Include a screenshot of your simulator running. [20 points]

2. Show the results of your open loop planner, that includes short description of your implementation, a screen shot of the virtual environment and your code. [20 points]

3. Show the results of your closed loop planner. Include the code, a short description and some screen shots. [20 points]

4. Include a short report of your hardware testing, including any code used and a discussion of vehicle performance. [30 points]

5. How would you improve the drift of the vehicle when it is following a path? Briefly discuss how this could be achieved. [10 points]