ME 133b - Lab 1 - Introduction to Quadrotors

Released: Wednesday 1/9/2019 Due: Friday 1/18/2019

This lab will show how to operate a typical quadrotor. Similarly to Lab 2 of ME133a, we will execute an open loop path and then compare the results with our intended path. This time, compared to the turtlebot robot used for the previous lab, the robot is much faster and it moves in 3D. This will create some difficulties we will address during the lab. This guide assumes some familiarity with ROS and its libraries. If you haven't done it already we encourage you to do the tutorials on the ROS webpage[1].

This lab will composed of two parts: in the first part we will execute the open loop commands of a previously recorded flight using the joystick. In the second part of the lab we will set the commands using a script (this part is very similar to lab 2 of ME133a). To check our code we will by two tools: one is a simple code to integrate our commands to check the shape of our intended path. The other is the simulator environment Gazebo, embedded in ROS. See the additional *Guide to Hector Quadrotor* for more details. These tools will allow us to test the code without the risk of crashing the vehicle. It will also allow us to develop and debug the code without using the limited hardware resources. To execute their code in the real platform the students will allocate a time slot with the TAs. Please read carefully the last section 'Deliverables' to know what you should do before, during and after the hardware demostration.

Open Loop Control for Quadrotor Bebop 2

We will be 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. The easiest way to start using the drone is by the the App Free Flight Pro, available for iPhone and Android. We will not using this type of control for this lab but it is important to keep in mind as a backup control, for example if the laptop runs out of battery and the quadrotor is in mid air.

We will connect to the Bebop, the Optitrack system and a keyboard for sending manual commands. Figure shows a schematic diagram with all the connections.
The following steps will be executed during lab. You don't have to pay a lot of attention but it will be helpful to understand better how to operate the robots for the project part of the class.

The first step is to connect the drone to your computer by running the bebop_driver, then we connect the joystick and we are ready to flight.

1. Turn on the drone
2. Connect to the drone WiFi. It will create its own network called 'BebopID' where ID is the id of that particular vehicle.
3. Launch the bebop ROS node

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

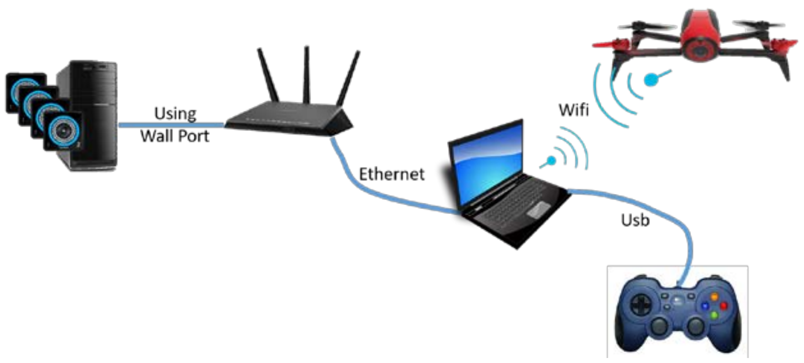


Figure 1: Schematic diagram during Lab 1.

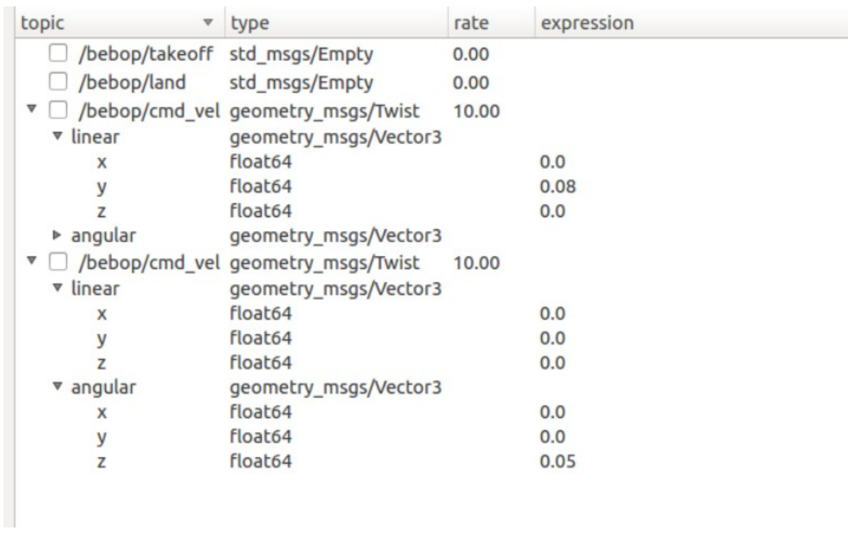
```
$ roslaunch bebop_driver bebop_node.launch
```

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

4. Connect the joystick
5. Launch the joystick ROS node

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

6. Take a short test flight to check the drone flights as expected.
7. An extra backup method is to send messages directly using the ROS message publisher. 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 7



topic	type	rate	expression
☐ /bebop/takeoff	std_msgs/Empty	0.00	
☐ /bebop/land	std_msgs/Empty	0.00	
▾ ☐ /bebop/cmd_vel	geometry_msgs/Twist	10.00	
▾ linear	geometry_msgs/Vector3		
x	float64		0.0
y	float64		0.08
z	float64		0.0
▸ angular	geometry_msgs/Vector3		
▾ ☐ /bebop/cmd_vel	geometry_msgs/Twist	10.00	
▾ linear	geometry_msgs/Vector3		
x	float64		0.0
y	float64		0.0
z	float64		0.0
▾ angular	geometry_msgs/Vector3		
x	float64		0.0
y	float64		0.0
z	float64		0.05

Figure 2: Predefined messages to operate the drone.

8. For this lab we will use Optitrack as ground truth. Start Optitrack node and check that it works.

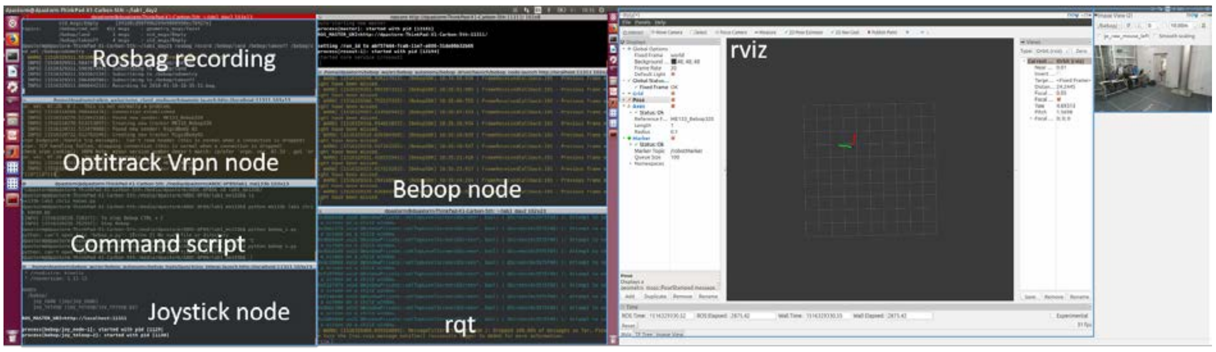


Figure 3: Typical screen configuration.

After we connect everything our screen should look similar to Figure. It is important to check the health of each node periodically to avoid using wrong data. Once we checked basic drone operation we will proceed to the two types of open loop control for this lab. The first part will record the joystick and then play it back while the second part will create the commands using a python script:

- **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. Besdies, be ready to kill all the communication with the drone (ctrl+c) and establish a new communication link using another computer or an smartphone.

  1. Put the drone in a flat area easy to identify. We are going to take off from the same spot later.
  2. Start recording the topics using rosbag record

     ```
         $ rosbag record -O _bagName_ /bebop/cmd_vel /bebop/land /bebop/
     odometry /bebop/takeoff /bebop/image_raw /vrpn_client_node/
     ME133 BeBop320/pose
     ```
  3. Flight around 1 min and then land.
  4. Stop recording by killing the terminal where rosbag was called with ctrl+c
  5. Check the new bag file with rosbag info. Register the number of messages for each topic in the report.

     ```
         $ rosbag info  bagName
     ```
  6. where _bag_Name_ is the name of the previously recorded bab. 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?

- **Scripted Open Loop:** execute the python script to command the drone your desired path. Before sending any movement commands, the drone has to take off. If you use the given template you should take off manually before executing your code. Assuming your are in the same folder as your script, you can run your code by running

```
$ python me133b_lab1.py
```

Is the drone doing what you expected? Why do you think it didn't follow exactly your path? What are the main differences with the Turtlebot robot we were using last term? You can repeat your path a couple of times, always being careful of not hitting anything or anyone.

Path Checker

To visualize your commands you can execute the python script called bebop$_s.py$.

```
$ python bebop_s.py
```

and then call your own script. Check that the rqt graph looks like Figure. This node will integrate your commands and create a rviz marker that will follow your commands. Open rviz in a new terminal and add a new object of type Marker as in Figure. Remember that the real drone will not follow these commands perfectly and we will cover better dynamic models and feedback control in next labs.

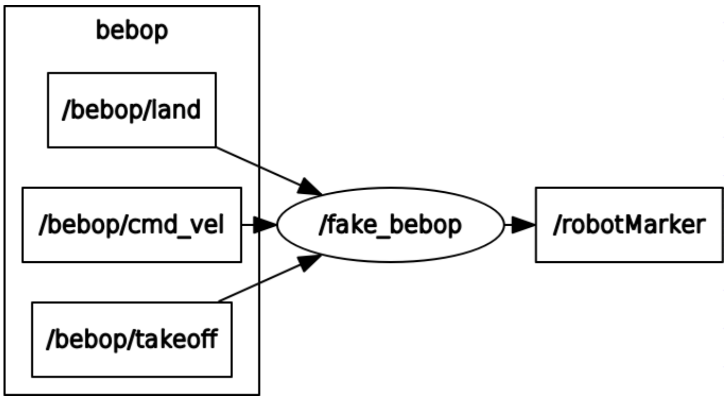
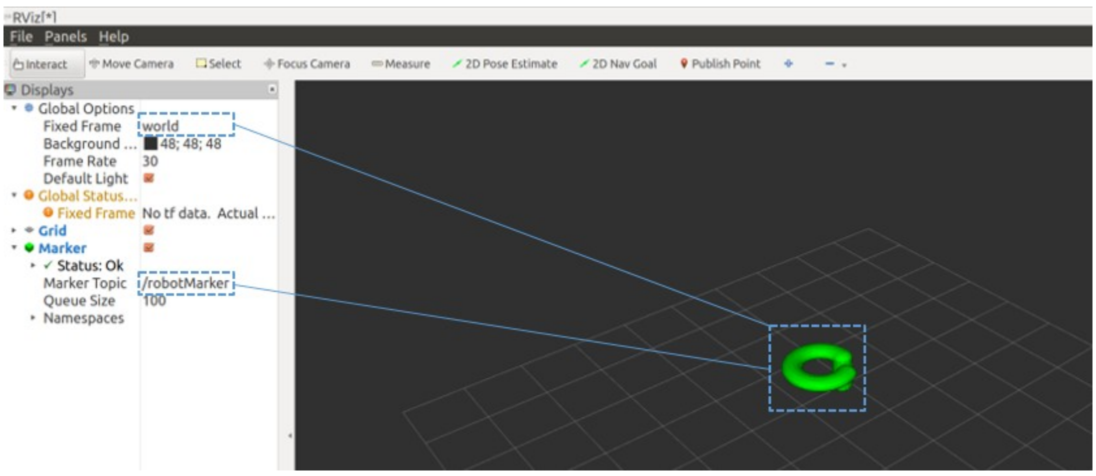


Figure 4: Path checker rqt graph. Check that all nodes are connected.


RViz[*]
File Panels Help
Interact Move Camera Select Focus Camera Measure 2D Pose Estimate 2D Nav Goal Publish Point
Displays

Global Options
Fixed Frame world
Background ... 48; 48; 48
Frame Rate 30
Default Light ✓
Global Status...
Fixed Frame No tf data. Actual ...
Grid ✓
Marker ✓
✓ Status: Ok
Marker Topic /robotMarker
Queue Size 100
Namespaces

Figure 5: Screenshot of Rviz during simulation. Check that the *Fixed Frame* and the *Marker Topic* are correct.

Homework Deliverables

For each lab work will be split in 3 categories:

1. before hardware demonstration:
   (a) Write an open loop planner following the template in the class webpage. We strongly recommend to follow the official tutorials for ROS to be able to understand the template. [20 points]
   (b) Optional: Test the code within the quadrotor simulator and take a screenshoot of the working code. [extra 10 points]
2. during hardware demonstration:
   (a) Command the quadrotor using the joystick and then play back the commands [30 points].
   (b) Execute your open loop planner in the quadrotor [10 points]
3. after hardware demonstration:
   (a) Include a short report of your hardware testing, including any code used and a discussion of vehicle performance. [30 points]
   (b) How would you improve the drift of the vehicle when it is following a path? Briefly discuss how this could be achieved. [10 points]