Autonomous Drone Capstone Project

Development Documentation and Notes

Matt DePero, Cole Hoffbauer,

Chris Mabe, Audrey Winzeler

Senior Capstone | Spring 2017

MIAMI UNIVERSITY

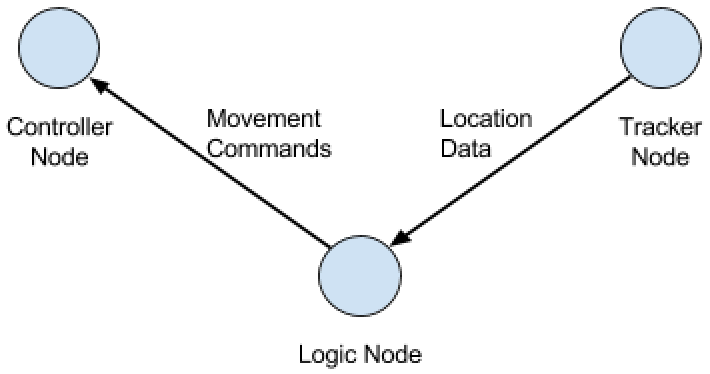
College of Engineering and Computing

ROS for Dummies

Overview

ROS is a tool that allows the modulor development of software packages and provides an easy to use interface for networking those nodes across a distributed system. In other words, it lets you create focused, self contained components to you robotic system and easily allow them to interact and work together. These modules are called **nodes** and the channels that they communicate on are called **topics**.

For example, in a simple robot system, you may have 3 nodes: one for controlling your robot, one for tracking your robot, and one for calculating the path of the drone. The tracking node **publishes** information about the location of the drone, the path calculating node **subscribes** to this information and then publishes movements commands from it, and the controller node subscribes to these commands and moves the robot accordingly.



This graph style system is at the core of ROS. You create (or download) nodes and each node publishes and/or subscribes to various topics. This provides full abstraction for each component of your system, and also makes it very easy to borrow tools from the open source community for use in your project.

ROS does this through a peer-to-peer network interface. A core process (called **Master**) is started which handles all messaging in the system. All messages are sent to this core process which handles routing it to the necessary nodes. This provides the abstraction to the programmer of being able to communicate directly with other nodes in the system. Because ROS is networked, nodes may be distributed over different hardware.

Development Environment

The logic of ROS is described above. This section describes the development environment for creating or editing custom nodes.

Filesystem

```
your_workspace_directory
↳ src
   →CMakeLists.txt
   ↳ your_package_name
        → CMakeLists.txt
        → package.xml
        ↳ src
             → your_code.cpp
             → your_code.py
```

Your workspace may contain more than one package in it. Each package can also contain more than one node (how your code gets compiled into executable nodes is defined in the CMakeLists.txt file of the package).

ROS usage is all about sourcing a setup.bash file. If you install ROS yourself, you will edit a file that makes your OS automatically source the ROS setup.bash file every time you open a terminal. The same goes for packages and nodes. For example, if you wanted to use a node with the source code described above...

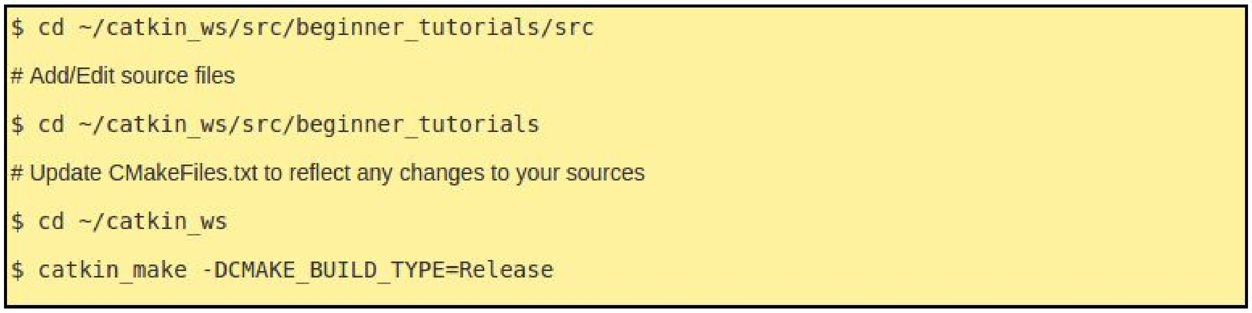
1. build/make the workspace
2. source the setup.bash file that gets created

Once you have your bash file sourced, you can simply execute your nodes via rosrun or roslaunch (if you need a ros file).

Hello World ROS

Only do the steps on this each individual page, don't move forward in the tutorials until this documentation tells you do. Also, these steps are not designed to actually create your first hello world ROS node (sorry), but rather to give you an understanding of how node development works and eventually be able to figure it out on your own.

1. Follow the instructions here
   - ➔ The instructions assume you have ROS installed. You must either be using the docker environment or install ROS. You likely don't need to source the setup.bash file if you installed correctly.
   - ➔ Pay attention to what distribution of ROS you have, don't just blindly copy and paste terminal commands
2. Follow the instructions here
   - ➔ Read the entire page. This tutorial explains in detail all the different components of a catkin package
   - ➔ This tutorial is where you are introduced to the details of how to actually build your workspace and then source the subsequent setup.bash file
   - ➔ Once you create your new package via the catkin create command, you must manually edit any dependencies that you did not include in the initial command by editing the package.xml file that gets created in the package folder
3. Read this page for more details on using the catkin workspace. The most important detail is in the yellow box (copied below) which outlines the big 3 steps necessary to create and build your custom packages



```
$ cd ~/catkin_ws/src/beginner_tutorials/src
# Add/Edit source files
$ cd ~/catkin_ws/src/beginner_tutorials
# Update CMakeFiles.txt to reflect any changes to your sources
$ cd ~/catkin_ws
$ catkin_make -DCMAKE_BUILD_TYPE=Release
```

   - ➔ After you catkin_make, you then need only to source the setup.bash file that gets created in *workspace/devel/setup.bash*.
   - ➔ Note before moving on that some 3rd party packages use rosbuild instead of catkin. Read this for cases where you have to use those.

4. One important part of the magic yellow box is updating your CMakeFiles.txt. See this page.
    - ➔ It is extremely dense. The most important takeaway is under Executable Targets which describes how you actually convert your .cpp files into the final output node.
    - ➔ The name of the executable specified in this CMake function is the same as the node that you eventually call with rosrun.
    - ➔ The default CMakeLists.txt file also contains the boilerplate for most all of these concepts
    - ➔ View our code in this project's github repo for example of what the wall of text on this page is talking about
    - ➔ Note that a project is the same as a package
5. Once you have built your code and sourced your setup.bash file, you can execute your node in the ROS system
    - ➔ $ roscore
    - ➔ $ rosrun <package_name> <node_name>
6. For a more concrete example of actually running code, view our tutorial on starting and running our my-cv node from the ground up.

If you've gotten this far in the intro to ROS and are still confused or I messed something up, contact Matt DePero (mattdepero@gmail.com) with questions.

Using Docker and Our Drone Challenge Image

[Things placed in brackets are optional, but are good ideas to do]

FOR LINUX MACHINES:
Oh hey! There's a video tutorial for this here!
To use the Drone Challenge image, you will first need to download Docker. This can be completed through following the steps on the Docker site. (https://www.docker.com/)

[Create a username and password for use on DockerHub in order to push your own images]

To get the Drone Challenge image our group has created, you will need to download it and build it yourself or pull it from the currently existing repo.

- To build it yourself, download the dockerfile and the other files that have been provided. Place it into a folder that you can find easily. Open the terminal, navigate to the directory you placed it in, and type "docker build -t (insert what you'd like to name it) . " (Make sure to include the ending period in the command, since this tells Docker the dockerfile you're working with is in the current directory)
- To use the current image, you will type "docker pull mabecr/drone_challenge:version2". This will pull the latest version of the image to your machine. Alternatively, you can pull a specific version from that repo by typing "docker pull mabecr/drone_challenge:(insert version name)".

To use to image you have downloaded, you will type "docker run -it --network="host" [--name (name for your container)] (image name)". This will start up an interactive (-i) container in TTY mode (-t) using your host machine's network (--network="host") with the name you gave it (name for your container). To exit the container once you are done working, type "exit" in the terminal.

To start an already existing container that has been stopped, [type "docker ps -a" to get a list of all of your existing containers. Find the one on the list you want to start], type "docker start -i (container name)".

Congratulations! You now have a working ROS/Bebop_autonomy environment you can begin developing with.

FOR WINDOWS MACHINES:
Lucky for you I took some time and made a video tutorial which can be found here!

In order to use a this image correctly with a windows machine, there are going to be a few extra steps.

First and foremost, you will have to go into your BIOS and enable "Hyper-V". Follow the tutorial here:
https://blogs.technet.microsoft.com/canitpro/2015/09/08/step-by-step-enabling-hyper-v-for-use-on-windows-10/

Download "Xming" (https://sourceforge.net/projects/xming/) and follow their installation instructions. You will not need putty for this, so feel free to exclude that from the installation. Make sure you install "XLauncher" however. You will need this to help set up your session.

To use the Drone Challenge image, you will need to download Docker for Windows. This can be completed through following the steps on the Docker for Windows site (https://docs.docker.com/docker-for-windows/install/#download-docker-for-windows).

Find an external wifi adapter. If you don't have one, the department will probably buy a cheap one for you. Plug it in, let the drivers for it install, and type in the search bar at the bottom of the start menu "Hyper-V Manager". This will allow you to customize the settings for the VM your containers run in. Click the Virtual Switch Manager button, click "new virtual network switch", click "External", and then "Create Virtual Switch". Name the switch whatever you want, and click the external wifi adapter in the drop-down menu, UNCHECK the box that says "allow management operating system to share this network adapter". Click apply.

Open your Windows Powershell, and type "ipconfig". Write down the IP that you are connected to the internet with. You will need this later. Click on your XLaunch icon, leave the first option as "Multiple Windows", "next", leave it as "Start No Client", "next", CHECK the box that says "No Access Control" (THIS WON'T WORK IF YOU DON'T). Click "next", save the configuration if you really want to, and then click "finish". You will get an icon in the tray at the bottom right if it worked that says "0:0".

To get the Drone Challenge image our group has created, you will need to download it and build it yourself or pull it from the currently existing repo.

- To build it yourself, download the dockerfile and the other files that have been provided. Place it into a folder that you can find easily. Open the Windows Powershell, navigate to the directory you placed it in, and type "docker build -t (insert what you'd like to name it) . " (Make sure to include the ending period in the command, since this tells Docker the dockerfile you're working with is in the current directory)
- To use the current image, you will type "docker pull mabecr/drone_challenge". This will pull the latest version of the image to your machine. Alternatively, you can pull a specific version from that repo by typing "docker pull mabecr/drone_challenge:(insert version name)".

We now need to tell the VM to use the switch. This is simply done by opening Hyper-V Manager after you have started Docker, right clicking the running machine, click settings, and add the switch we made before. Double check it is using the correct wifi adapter, as you don't want it to be your machine's primary adapter.

Now that we have our VM configured, XMing running, and the image ready, we can spin up our container. Type "docker run -it -e DISPLAY=<The IP you wrote down earlier>:0.0 --network="host" [--name container name] (image name). This will create a container that is interactive (-i) in TTY mode (-t) with and environment variable called "Display" set to your Xforwarded display using the host network (--network="host") with the name you entered.

[A good idea to test your Xforwarding is to create an image that downloads and installs firefox, start the container in the above way, and type "firefox". If you can see the gui, you did it correctly.]

Congratulations! You now have a working ROS/Bebop_autonomy environment you can begin developing with.

Installing and Running the Gazebo Simulation

Some notes before you begin:

- At the time I am writing this, no model exists for the Parrot Bebop drone. As a result, this tutorial uses a model called Hector Quadcopter, which is similar to the Bebop.
- This tutorial is written for Linux. Gazebo, the simulation environment, will not run correctly on Mac or Windows. It could be possible with docker, but we have not investigated that option.
- This section of the tutorial assumes that you have ROS running already, so make sure you have that set up before you begin.
- Part of this tutorial involves setting up LSD slam in order to use for mapping the environment. We ran into some problems with LSD slam, so you may want to replace it with a different package, such as ORB slam 2 (https://github.com/raulmur/ORB_SLAM2). I did not get around to incorporating ORB slam with the simulation, but it showed more promise that LSD slam, and was fairly simple to run.

Creating the environment:

1. Create a catkin workspace if you have not done so already, or would like to use a new one for this project (which I highly recommend).
2. Install opencv2 using instructions from http://docs.opencv.org/2.4/doc/tutorials/introduction/linux_install/linux_install.html, and libQGLViewer using instructions from http://libqglviewer.com/installUnix.html. You may also need to install additional dependencies for these packages.
3. Install hector_quadrotor using instructions from the “Prerequisites” section of the documentation. Use Indigo instead of Hydro as the ROS version. (http://wiki.ros.org/hector_quadrotor/Tutorials/Quadrotor%20indoor%20SLAM%20demo). If you have issues with this installation, use hector_quadrotor.rosinstall instead of tutorials.rosinstall.
4. Git clone the repository for hector slam (https://github.com/tu-darmstadt-ros-pkg/hector_slam.git) into the workspace. I had issues using hector_quadrotor without this package.
5. Install dependencies according to lsd-slam documentation in the Installation section (https://github.com/tum-vision/lsd_slam). Make sure you do the commands under the indigo section, not fuerte. Clone the repository according to the instructions.

Running the environment:

1. Change directory to the workspace if you are not already there.
2. In one terminal window: “source devel/setup.bash” (sets setup.bash as source, as described in Matt’s ROS tutorial).
3. In the same window: “roslaunch hector_quadrotor_demo indoor_slam_gazebo.launch” (starts up simulation, including roscore).
4. In a second terminal window, use this command to add the lsd slam package to the path: “export ROS_PACKAGE_PATH=$ROS_PACKAGE_PATH:~/<workspace_name>/src/lsd_slam/” (you will need to do this every time you run the lsd slam node).
5. Compile using “rosmake lsd_slam”.
6. Run lsd_slam_viewer using “rosrun lsd_slam_viewer viewer”. This will start a new node that shows a map of the environment.
7. In a third terminal window: compile lsd slam again, using steps 4 and 5.
8. Run lsd_slam using “rosrun lsd_slam_core live_slam /image:=/front_cam/camera/image /camera_info:=/front_cam/camera/camera_info”. This will run another node that shows the view from the camera, with additional output from lsd slam.
9. In a fourth terminal window use this command so you can control the drone using the keyboard: “rosrun teleop_twist_keyboard teleop_twist_keyboard.py”. You may need to install teleop_twist_keyboard, which can be installed using “sudo apt-get install ros-indigo-teleop-twist-keyboard”.
10. In a fifth terminal window, use the following command to record data to a rosbag: “rosbag record /lsd_slam/graph /lsd_slam/keyframes /lsd_slam/liveframes -o <filename>.bag”
11. In order to play back the rosbag, compile lsd slam using steps 4 and 5. Then, use the command “rosrun lsd_slam_viewer viewer <filename>.bag”.
12. If you are asked to reindex a bag, use “rosbag reindex <bagname>.bag”.

This is as far as I managed to get (concretely) with the simulation. If you have any questions, contact Audrey Winzeler at awinzeler@gmail.com.