AER1217: Development of Unmanned Aerial Vehicles
Lab 1: Introduction to ROS

1 Introduction

ROS is an abbreviation for Robot Operating System. ROS is an open source middleware for software management, communication, and process handling. It is widely used in robotics with a large community of users. Please refer to here for full documentation, tutorials, and examples.

The laptop used in this course runs **Ubuntu 16.04** LTS (64bit) and has ROS **Kinetic** distribution (or known as *distro* in short) installed on it through the recommended *desktop-full install*. The laptop will serve as a ground control station for flying the **Parrot AR Drone 2 Power Edition** in a VICON motion tracking environment in the Indoor Robotics Lab (UTIAS room 195). The required ROS packages for the purpose of this course are installed on the laptop, which include but not limited to **gazebo**, **vision_opencv**, **ardrone_autonomy**, **ardrone_tutorials**, and **vicon_bridge**.

This document will briefly mention some "prerequisites" required to perform the labs and project, and provide useful links to tutorials which are intended for self-learning. The document will go into more course-specific details in terms of installing ROS packages and using certain ROS packages/features.

2 Software and Packages used in AER1217

The software required for this course is listed below, and should be installed automatically by following the automated setup scripts (assuming Ubuntu 16.04 has been installed). The following serves as a checklist for reference.

- Ubuntu 16.04 (Xenial)
- ROS Kinetic Kame
- Gazebo (version 7.0)
- Python 2.7.x
- OpenCV 3.3.x (pre-installed with ROS)

3 Prerequisites

3.1 Using Bash Terminal in Ubuntu

Ubuntu terminals run using Bash scripting. If you are not familiar with using Bash, please go through the tutorials on this website.

3.2 Programming in Python

ROS supports both **C++** and **Python** as programming languages. Python is the *recommended* programming language because it's simpler and less time-consuming to implement, as code written in Python do not need to be re-compiled before running. The AR Drone tutorials and examples are written using Python, making code modifications more convenient. Most Python codes written in the **ardrone_tutorials** package are in Python version 2.7.x (not 3.0).

To ensure that the course is balanced well between development of the AR Drone and learning a new programming language, most labs will have Python codes ready to be run with slight modifications required. If you have experience programming with C/C++/MATLAB, Python should be an easier programming language to learn and use. Nevertheless, a quick introduction to Python syntax can be found here.

To perform any mathematical operations in Python, you will use the NumPy package (pre-installed with ROS). For those familar with Matlab, NumPy arrays have almost the same behaviour and similar syntax as Matlab arrays/matrices. For a quickstart tutorial on using NumPy, follow the link here.

A quick list of numpy equivalents for common Matlab commands can be found here. Out of the many differences between both languages, there are three important ones to be aware of:

1. In order to correctly create a row vector in Python for matrix operations, two nested brackets must be used to specify a 2D array. Below is an example of the matrix transpose working incorrectly for a 1D array.

```
> x = np.array([[1,2,3]])
> x.T
array([[1],
       [2],
       [3]])
> y = np.array([1,2,3])
> y.T
array([1, 2, 3])
```

```
>> x = [1,2,3];
>> disp(x')
     1
     2
     3
```

Figure 1: Matrix transpose example (left: Python commands, right: Matlab commands)

2. Similar to almost every other programming language out there, Python array indexing begins at 0 and excludes the last index, whereas Matlab array indexing begins at 1 and includes the last index. Python indexing also uses brackets, whereas Matlab uses parenthesis.

```
> x = np.array([[1,2,3]])
> x[0,0:2]
array([1, 2])
```

```
>> x = [1,2,3];
>> disp(x(1,1:2))
     1     2
```

Figure 2: Array indexing example (left: Python commands, right: Matlab commands)

3. Numpy arrays by default pass by reference whereas Matlab arrays pass by copy. This is an example of how mutable data structures are handled in Python. To pass by copy, the 2nd line in Python should be `y = np.copy(x)`.

```
x = np.array([[1,2,3]])
y = x
x[0,0] = 10
print(x); print(y)
[[10  2  3]]
[[10  2  3]]
```

```
>> x = [1,2,3];
>> y = x;
>> x(1) = 10;
>> disp(x); disp(y);
    10     2     3

     1     2     3
```

Figure 3: Equating arrays example (left: Python commands, right: Matlab commands)

The Python files in the **ardrone_tutorials** package can also be easily modified to achieve what is needed for the purpose of this course, which can be found in this Github repository within the **/src** folder .

3.3 Using OpenCV Libraries

The installed ROS comes with the **vision_opencv** package, which uses the OpenCV 3 libraries as described here. An example of image processing algorithm that will be used in the labs is colour thresholding, and the documentation on the various thresholding methods can be found here.

4 Installation

4.1 Installing Ubuntu 16.04

You will need to have Ubuntu 16.04 installed in your computer for this course. If you have an existing operating system (Windows), we would recommend that you dual boot your laptop. Running a virtual machine or Docker image instead may have issues with performance. If dual booting, make sure that you allocate at least 15GB of disk space as ROS is resource heavy. The following tutorial provides one way of dual booting Ubuntu 16.04 and Windows.

Before you install Ubuntu 16.04 on your computer, make sure you meet the following requirements:

- A USB disk of capacity around 4GB
- Backup your Windows data (optional)

After meeting the requirements, start by downloading the Ubuntu 16.04 from Ubuntu 16.04 (Xenial). Download `64-bit PC (AMD64) desktop image` in the `Desktop image`.

Next, download Universal USB Installer, which is a Live Linux Bootable USB Creator that allows you to choose from a selection of Linux Distributions to put on your USB Flash Drive from Universal USB Installer.

Insert the USB drive and open the Universal USB Installer and choose `continue`. On the Setup your Selections Page, choose `Ubuntu` in Step 1, and browse your computer to select the Ubuntu*desktop*.iso you downloaded in Step 2. Then, choose the drive letter of your USB in step 3 and check the box to `Fat32 format USB Drive`. Click `Create`. After the installation process, click `close`.

In the search bar (in Windows), type `Create and format hard disk partitions` and open it. Right click the disk that you want to shrink, and choose `shrink volume`. In the shrink interface, choose the allocation space for Ubuntu in `Enter the amount of space to shrink in MB`. You should allocate at least 15GB of disk space, but you cannot allocate the space more than size of available shrink space. After choosing a right amount of space to shrink, click `Shrink`.

Then, restart your computer and go to BIOS menu to allow booting from USB. Every computer is different from how to open the BIOS menu, but most computers use F10 or F12. After booting, click `Install Ubuntu`. In `Installtion type`, select the option `Something else` and select the partitions we provided in step 3 (around 15000 MB). A detailed installation process can be found online (for example here).

4.2 Installing ROS on your laptop

We have automated most of the ROS download through the script uploaded here. Follow the steps below to download and run the scripts, note that this installation will take some time to complete. While waiting feel free to read the next section to find out what the script is doing under the hood. This will give you a deeper understanding of how ROS is setup. In short, the scripts are installing ROS and the relevant packages for this course, as well as setting up the file environment.

Start by opening a new terminal run the commands below. They will install git and clone the download scripts from the aer1217 repository.

```
sudo apt-get install -y git
cd $HOME
sudo git clone https://bitbucket.org/aer1217/setup_script.git
cd setup_script
```

Now set the permission to enable script file execution, using the following lines in Terminal:

```
sudo chmod +x setupscript_1604.bash
sudo chmod +x usersetup_1604.bash
```

To run the script, type the following in Terminal:

```
./setupscript_1604.bash
```

Close the Terminal window and **relaunch a new Terminal**, then in the new Terminal, go to the setup_script directory and type:

```
./usersetup_1604.bash
```

The script will prompt for your password for the very first time, and use it to perform commands that require admin privileges. If you encounter permission error when launching ROS packages, try typing `sudo rosdep fix-permissions` in Terminal and report the errors to the TAs.

Gazebo is a robotics simulation tool useful for testing custom robots in custom environments. We will use Gazebo to simulate the ARDrone. Gazebo will already be installed by the scripts above. However, to obtain more packages and models, you may proceed with the next command. Note that this would install Gazebo

from scratch as well, but in our case, it will just update any missing packages.

```
sudo apt-get install ros-kinetic-gazebo-ros-pkgs ros-kinetic-gazebo-ros-control
```

Another useful tool to have is pip, which is a program that installs Python libraries. You can install pip by running the following command:

```
sudo apt-get install python-pip
```

4.3 Testing the Installation

The files which have been installed are located in the `aer1217/` folder. Close your Terminal and reopen it. When Terminal launches, it will execute the `.bashrc` file which will source the packages within the `aer1217/extras` folder. One of the packages is an ARDrone simulator, you can run it using the following command:

```
roslaunch aer1217_ardrone_simulator ardrone_simulator.launch
```

If your installation was successful, you should see Gazebo open in an empty world with the ARDrone on the ground. Click play at the bottom of the screen, the ARDrone should takeoff and begin to hover with some drift.

To land the ARDrone, you can run the following command in a new terminal:

```
rostopic pub -1 /ardrone/land std_msgs/Empty
```

To takeoff the drone again, you can run the following commmand:

```
rostopic pub -1 /ardrone/takeoff std_msgs/Empty
```

What you have just done was publish a ROS message to a topic, you can read more about this in the next section or online in the ROS documentation. We will also go over this in tutorial.

To close the simulator, press ctrl-c in the terminal running the launch file.

5 Robot Operating System (ROS)

Treat this section as documentation for the fundamentals of ROS. Since ROS is such a complex tool we do not expect you to understand all its details. We will run a tutorial to help you digest this information as well as update the documentation as the course continues, so don't panic!

This section describes how to install ROS without using our automated scripts as well as details about its architecture. Since the installation has been automated by our scripts, you do not need to run the commands in this section.

5.1 Installing ROS Without Automated Script

The instructions at this link installs ROS Kinetic and its dependencies. Our script `setupscript_1604.bash` runs all the commands in that set of instructions.

5.2 Setting Up Catkin Workspaces

ROS Packages are modified and built inside a Catkin Workspace, you can refer to here for additional documentation. We have setup a catkin workspace for you in the script `usersetup_1604.bash`. The workspace is setup in the `aer1217/extras` directory by following the steps below.

```
cd
mkdir aer1217 && cd aer1217
```

```
mkdir extras && cd extras
mkdir src && cd src
catkin_init_workspace
```

Once the catkin workspace is made, we need to source the workspace so that ROS can find our packages the next time a terminal opens.

```
. extras/devel/setup.bash
echo "source ~/aer1217/extras/devel/setup.bash" >> ~/.bashrc
```

Finally, we install certain dependencies required for our packages using the following commands.

```
sudo apt-get install daemontools libudev-dev libiw-dev
sudo apt-get install libsdl1.2-dev
```

Note: In the process of installing ROS packages, it is common to encounter error messages saying that certain dependencies are not found. Observe the logs within the Terminal and search online on how to obtain that specific package or dependency required. Alternatively, type `rosdep install --from-paths src --ignore-src --rosdistro kinetic -y` in the Terminal. This command will install any dependencies that are needed by the packages your workspace.

5.3 Installing ROS Packages

There are two methods to install ROS packages:

1. **Installing Debian Packages**

   If a Debian package is available, it is usually named as *ros-<distro>-<stack_name>*. Note that in some cases, the *<stack_name>* is not necessarily the package name, shown in Figure 4, where the **rxparamedit** package is within the **freiburg-tools** stack. In Ubuntu terminal:

   ```
   sudo apt-get install ros-groovy-freiburg-tools
   ```

   where we replaced the underscores (_) with dashes (-). This package is used just for the purpose of showing an example, we do *not* need this package installed for the labs or project.

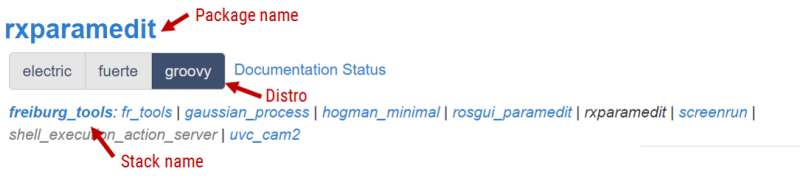


Figure 4: An example: installing **rxparamedit** Debian package in ROS

2. **Cloning From Source**

   When the Debian package is not available, we can install from sources. Most source codes of ROS packages are hosted online on Github; installing them involves downloading the packages and compiling them on our own computer. We will use the **ardrone_autonomy** and our custom tutorial package **ardrone_tutorials** as an example to show the steps involved.

   ```
   git clone https://github.com/AutonomyLab/ardrone_autonomy.git
   ```

```
git clone https://bitbucket.org/aer1217/ardrone_tutorials.git
```

Once the source packages have been copied, we need to build its code by running `catkin_make` in our workspace `aer1217/extras`, so we run the following commands.

```
cd ..
catkin_make
```

5.4 Understanding ROS: The Tutorial Package

The ROS wiki page contains a comprehensive list of tutorials that can be very helpful in understanding the core components of ROS and how ROS works. Note that not all will be used throughout the course; the important ones will be covered in the remaining of this document.

5.5 Writing Codes and Including Them in Package

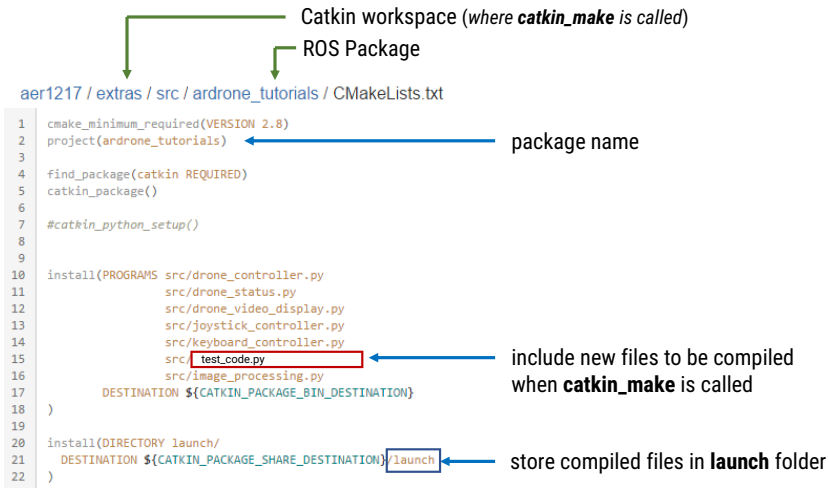


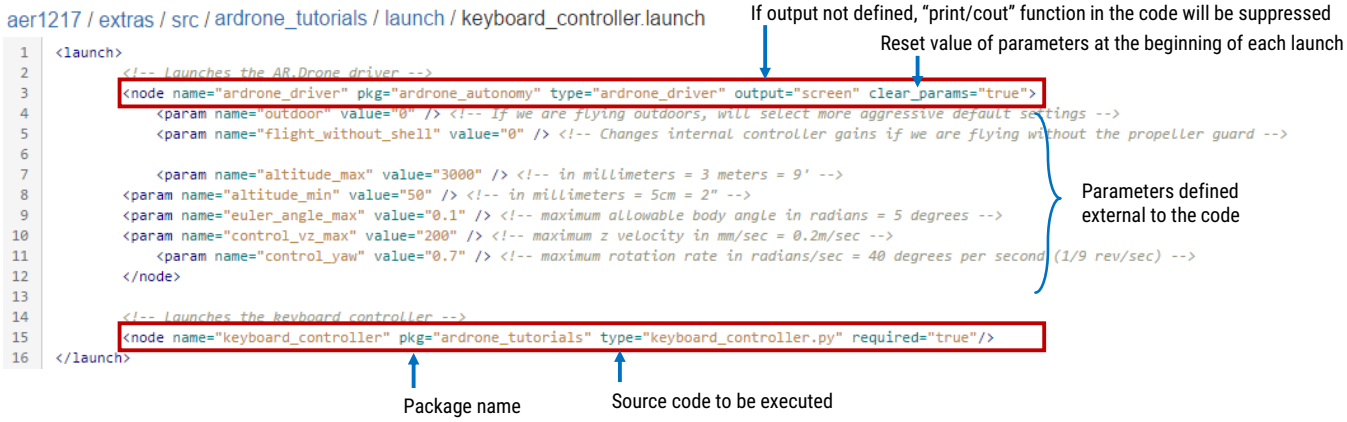
Figure 5: Using CMakeLists.txt in ROS

When working on the labs or project, please do not modify the code that were provided from the source. If needed, create a copy of the original code and make the necessary modifications. This new file, for example, called `test_code.py`, is to be included in the **CMakeLists.txt** under the **ardrone_tutorials** (package) folder.

As shown in Figure 5, **CMakeLists.txt** files contains a list of dependencies and/or classes that the package requires to function properly. When the `catkin_make` (compile) Terminal command is run in the workspace (in this case, folder of `~/aer1217/extras/`), catkin will look into all the **CMakeLists.txt** files within the workspace and compile the listed codes accordingly.

To launch the new code (known as *node*), we have to include the .py code into the launch file, found under the `<package_name>/launch` folder. There are several .launch files in the **ardrone_tutorials** package that can serve as good templates.

The launch file can contain multiple files (*nodes*) from different packages to be run together. For example, as shown in Figure 6, the `keyboard_controller.launch` file launches the compiled `ardrone_driver.cpp` code (*type*)




```
1  <launch>
2      <!-- Launches the AR.Drone driver -->
3      <node name="ardrone_driver" pkg="ardrone_autonomy" type="ardrone_driver" output="screen" clear_params="true">
4          <param name="outdoor" value="0" /> <!-- If we are flying outdoors, will select more aggressive default settings -->
5          <param name="flight_without_shell" value="0" /> <!-- Changes internal controller gains if we are flying without the propeller guard -->
6
7          <param name="altitude_max" value="3000" /> <!-- in millimeters = 3 meters = 9' -->
8          <param name="altitude_min" value="50" /> <!-- in millimeters = 5cm = 2" -->
9          <param name="euler_angle_max" value="0.1" /> <!-- maximum allowable body angle in radians = 5 degrees -->
10         <param name="control_vz_max" value="200" /> <!-- maximum z velocity in mm/sec = 0.2m/sec -->
11         <param name="control_yaw" value="0.7" /> <!-- maximum rotation rate in radians/sec = 40 degrees per second (1/9 rev/sec) -->
12     </node>
13
14     <!-- Launches the keyboard controller -->
15     <node name="keyboard_controller" pkg="ardrone_tutorials" type="keyboard_controller.py" required="true"/>
16 </launch>
```

Figure 6: Using .launch file in ROS

in the **ardrone_autonomy** package (*pkg*) and the `keyboard_controller.py` code (*type*) in the **ardrone_tutorials** package (*pkg*). The node can contain multiple parameters (*params*) that allow the users to modify the value of certain variables without having to recompile the code using the `catkin_make` Terminal command. If we create a new Python file `test_code.py`, we will have to include it into the .launch file by modifying line 15 in Figure 6 with:

```
<node name=...  pkg=...  type="test_code.py" .../>
```

With the launch file properly defined, use the command in the Terminal:

```
roslaunch <pkg_name> <launch_file>.launch
```

to launch the package with their corresponding nodes. In the example given in Section 5.3, we call `roslaunch ardrone_tutorial keyboard_controller.launch` in the Terminal.

Multiple `roslaunch` can be executed in different new Terminal window. To obtain drone position and attitude, the **vicon_bridge** package will be used to obtain data from the VICON motion capture system. To launch that package (installed on the course-specific laptop), use `roslaunch vicon_bridge vicon.launch` in a new Terminal.

For more information on launch file, refer here.

Note: `rosrun` launches a single node, while `roslaunch` can launch multiple nodes.

5.6 ROS Topics: Subscribing and Publishing

Different nodes across different packages can communicate and share information in the form of ROS messages through `rostopic`. While ROS is running with active nodes launched, the list of topics published by the nodes can be seen using `rostopic list` command in the Terminal. Each topic can have multiple publishers and subscribers; this can be seen using `rostopic list -v` (verbose) command.

The type of message within each topic can be identified using `rostopic type <topic_name>`, followed by `rosmsg show <message_type>`. The content of each topic can be printed out in real time using `rostopic echo <topic_name>`. For example, when the **ardrone_simulator.launch** is launched as described in section 4.3, we can open a new Terminal window, and observe the vicon data of the AR Drone using

```
rostopic echo /vicon/ARDroneCarre/ARDroneCarre.
```

More information on rostopic can be found here.

One important topic in the **ardrone_autonomy** package that we will be using in the visual-based navigation is

`/ardrone/front/image_raw`,

which contains the RGB image data from the drone's front camera.

In Python **drone_video_display.py**:

```
rospy.Subscriber("/ardrone/front/image_raw",Image,self.ReceiveImage)
```

"`ReceiveImage(self,data)`" is a function that will be triggered whenever the subscriber receives a new image file. The image file will be stored in the format of ROS image, which can be converted to OpenCV image using the CV Bridge library:

```
cv_image = self.bridge.imgmsg_to_cv2(<ROS Image>, desired_encoding="bgr8")
```

Once the image is converted to the OpenCV format, we can utilize the image processing libraries as described in Section 3.3.

Similarly, the position and orientation of the AR Drone, according to the VICON motion capture system, can be subscribed to using the topic `/vicon/ARDroneCarre/ARDroneCarre`. Of course, the **vicon_bridge** package has to be installed similar to the procedures described in Section 5.3, the AR Drone with visual markers has to be calibrated and registered with the VICON system, and this vicon node has to be launched using `roslaunch vicon_bridge vicon.launch` before the topic can receive data. This is simulated for you if you are running the Gazebo simulator, but you will need to launch this node for the real experiment.

If we want to control the AR Drone through a Python script instead of controlling it using the terminal, for example like taking off in section 4.3, we can publish to the `/ardrone/takeoff` topic within a Python script file. The variable types that AR Drone takes are described here.

For more information on writing publishers and subscribers using Python, see here.

5.7 ROS Services

For certain actions that terminate quickly, ROS service is preferred over ROS topic. One service that we will be dealing with in this course is the `/ardrone/setcamchannel`. Setting the channel to 1 activates the forward-looking camera, and channel 0 activates the bottom-facing camera.

In Terminal, ROS service can be called:

```
rosservice call /ardrone/togglecam     or
rosservice call /ardrone/setcamchannel 0
rosservice call /ardrone/setcamchannel 1
```

In Python code, ROS service can be called through this code snippet:

```
rospy.wait_for_service("ardrone/setcamchannel")
try:
    switchcam = rospy.ServiceProxy("ardrone/setcamchannel",CamSelect)
    switchcam(1) # for forward facing, switchcam(0) for downward facing
except rospy.ServiceException, e:
    print "Service call failed:  %s"%e
```

Due to hardware limitations, it is not possible for the AR Drone 2 to stream images from two cameras at the same time. (And toggling between camera channels in code at high frequency could lead to undesirable effects where image data from the front camera is thought to be from the bottom camera – please toggle slowly).

Note: the differences and when to use topics/services are discussed here.

5.8 AR.Drone ROS Package

Full documentation on the topics and services on the **ardrone_autonomy** package can be found here.

6 Deliverable and Grading

This lab is worth 5%. Please demonstrate that at least one computer in each group has simulation environment successfully installed on **January 25** during **Lab 1**.

- Demonstration of successful installation of ROS simulation environment, including AR.Drone *takeoff* and *landing*.
- Demonstration of ROS commands, including `rostopic`, `rosservice`, `rosrun`, etc.