# Module 1 - ROS


**Lesson Objectives:**

1. Learn fundamental concepts of ROS.
2. Develop basic operational understanding of ROS through application.
3. Design, build, and test a basic chat client using ROS.

**Agenda:**

1. ROS Introduction.
2. ROS Jupyter Notebook.
3. ICE1 Jupyter Notebook.

## 1 ROS Introduction.

**Robotics Operating System (`https://www.ros.org/about-ros/`):**

The Robot Operating System (ROS) is a flexible framework for writing robot software. It is a collection of tools, libraries, and conventions that aim to simplify the task of creating complex and robust robot behavior across a wide variety of robotic platforms. ROS is sometimes called a meta operating system because it performs many functions of an operating system, but it requires a computer's operating system such as Linux.

Why? Because creating truly robust, general-purpose robot software is hard. From the robot's perspective, problems that seem trivial to humans often vary wildly between instances of tasks and environments. Dealing with these variations is so hard that no single individual, laboratory, or institution can hope to do it on their own.

As a result, ROS was built from the ground up to encourage collaborative robotics software development. For example, one laboratory might have experts in mapping indoor environments, and could contribute a world-class system for producing maps. Another group might have experts at using maps to navigate, and yet another group might have discovered a computer vision approach that works well for recognizing small objects in clutter. ROS was designed specifically for groups like these to collaborate and build upon each other's work, as is described throughout this site.

**Noetic Ninjemys (`http://wiki.ros.org/noetic`):** *Is the 13th ROS distribution release and is the distro we will be using in this course. Previously, this course used ROS Melodic. Noetic is supported until May 2025 and includes Python3 integrations. Existing capstones and research have transitioned to Noetic, and sticking to the same distro allows for better collaboration*

**Catkin (`http://wiki.ros.org/catkin`):** *Think of this as a compiler. It is a low-level build system that provides macros and infrastructure for ROS. When doing research on projects or looking at other tutorials, you may find references to rosmake or roscreate-pkg; Ignore those and use catkin_make and catkin_create_pkg. Not important at this time, but just something to keep in mind*

**Master (`http://wiki.ros.org/Master`):** *The ROS Master enables all other components in the system to communicate. It is the controller of the network. More on this later*

**Packages (`http://wiki.ros.org/Packages`):** *Software in ROS is organized in packages. You can think of a package as a component in the overall system. For example, we may have a package called ground_robot or one called drone. You might have multiple ground_robots in your system, but the ground_robot package will provide common functionalities among them.*

**Package.xml:** *Each package contains a manifest named package.xml that describes the package in the Extensible Markup Language (XML) format. In addition to providing a minimal specification describing the pacakge, the manifest defines properties about the package such as the package name, version numbers, authors, maintainers, and any dependencies on other packages.*

**Nodes (`http://wiki.ros.org/Nodes`):** *A node is an executable (Python in this course) that uses ROS to communicate with other nodes. Nodes communicate with each other using topics. An example might be a node that reads distance information from a LIDAR, publishes that information to a topic, and then a controller node might use that information to move the robot accordingly to avoid obstacles*

**Topics (`http://wiki.ros.org/Topics`):** *You can think of a topic as a named bus or communication medium over which nodes exchange messages. An example topic might be current_location which a node could use to share the ground_robot's current location with a controller. The controller does not care what type of location source is providing the current information (it could be a Bitcraze Loco Positioning tag, a Camera Vision System, or outdoor GPS system) as long as the topic's formatting is maintained. That's a huge benefit gained when working with multiple nodes.*

**Messages (`http://wiki.ros.org/Messages`):** *Nodes comunicate with each other by publishing messages to topics. A message is a simple data structure, comprising typed fields. There are many prebuilt messages for example, the Point message contains float64 x, float64 y, and float64 z values. Custom messages can be*

*created using a msg file and are stored in the msg subdirectory of a package.*

**Publisher and Subscriber:** *The publisher and subscriber concept is a way for nodes to communicate messages on certain topics. A publisher is the node that writes messages to a topic, while the subscriber reads messages from the topic. For example, a LIDAR node might publish distance information to a topic while a Controller node would subscribe to that message and take action every time the topic is updated. You can have multiple subscribers to the same topic. You can also have multiple publishers to the same topic, but realize there is no way for the subscriber to know where the message is coming from. So you may have a LIDAR and a IR sensor both publishing on a distance topic, but one may be more accurate so there would be no way to tell*

**Services:** (`http://wiki.ros.org/Services`) *A service provides a request response protocol but is different than a topic. A providing ROS node (Server) offers a service under a string name and a client calls the service by sending the request message and awaiting the reply. An example would be if you provide a point A and point B to a provider and then that provider plans a path and returns back the path to the client*

## 2 ROS Jupyter Notebook.

We will use a Jupyter Notebook to practice and provide more insight into the terms above.

1. On the master, open the Jupyter Notebook server:

```
dfec@master:~$ roscd usafabot_curriculum/Module1_ROS
dfec@master:~$ jupyter-notebook
```

2. Open the ROS Jupyter Notebook, "ROS.ipynb", and follow the instructions within the notebook.

## 3 ICE1 Jupyter Notebook.

The ICE1 Jupyter Notebook will guide you through implementation of a chat client using ROS.

1. On the master, open the Jupyter Notebook server (if it is not already open):

```
    dfec@master:~$ roscd usafabot_curriculum/Module1_ROS
    dfec@master:~$ jupyter-notebook
```

2. Open the ICE1 Jupyter Notebook, "ICE1_Talker.ipynb" and follow the instructions within the notebook.

**Checkpoint. Take a screenshot or show the instructor the following:**

1. List of running topics.
2. The *rqt_graph* for the nodes and topics currently running.
3. An echo of messages sent via the chat topic.
4. Show what type of message is sent over the chat topic (Hint: use the rostopic command).

## 4 Assignments.

☐ (Before lesson 3) Go to `http://wiki.ros.org/ROS/Tutorials` on your master and work through ROS tutorials 2-16 on the master (remember, you only need to look at Python tutorials). This shouldn't take more than two hours.

## 5 Next time.

- Lesson 3: Quiz on ROS
- Lesson 4: Module 2 - Linux for ROS