Introduction to ROS

(Robot Operating System)

What is ROS?

- ROS, contrary to its name, is more of a framework than an OS.
- ROS runs on top of Linux to provide interprocess communication, resource allocation, and hardware communication and abstraction.
- Put simply, ROS gives you an easy way to network multiple programs together while also distributing system resources automatically.

Getting Started

- Assuming ROS is installed by this point, the next step is to set up our environment.
- Type "printenv | grep ROS" into your command terminal. If you see any results then your environment variables are set up properly.
- If you did not see anything, type "source /opt/ros/<version>/setup.bash". Replace <version> with whichever distribution of ROS you are using (Indigo, Jade)
- To simplify things for future make use of .bashrc

Creating a ROS Workspace

- In ROS projects are contained within a workspace folder that ROS must know about in order to properly build and run your programs.
- To create a ROS workspace, type the commands below into your command terminal
- mkdir -p ~/newbie_ws/src
- cd ~/newbie_ws/src
- catkin_init_workspace

ROS Filesystem Tools

- rospack - Lets you see information about ROS packages.
- roscd - lets you change working directorys to a given ROS package or directory.
- roscd log - same as roscd but will always take you to the directory in which log files for a package are stored.
- rosls - lets you see all files and subfolders in a given ros package.

Creating a ROS package

- Change your directory to the location of the src folder of your workspace (ie. cd ~/<workspace name>/src)
- Type this command in your terminal
  - catkin_create_pkg <package name> std_msgs rospy roscpp
  - The general format of this command is shown below
  - catkin_create_pkg <pkg name> <depend1> <depend2> ... <depend n>
- Your workspace/src folder should now contain a CMakeList.txt file, and a folder for your package.

What makes a ROS package?

- Every ROS package needs a few things
  - An overall CMakeList.txt inside your src folder. This tells ROS about what packages are in your workspace.
  - A package specific CMakeList.txt. This tells ROS how to find build and run dependencies and is used in building your package.
  - A package.xml file. This tells ROS version information, package name, maintainers, and dependency information.

- Take a look at each of these files to see what information they contain to help ROS know how to handle your packages.

Building a Package

- Now let's go ahead and build your first package.
- To do this, type these commands into your terminal:
  - cd ~/<workspace name>/
  - catkin_make
- In general to build ALL packages in a workspace, you have to be in your workspace root (eg. ~/<workspace name>), and you must then run catkin_make.
- You can also build specific packages, and set build flags with
  - catkin_make <package1> <package2> ... <package n> <flags>

When do I get to program :(

- I know, 6 whole slides on file system manipulation. I've said it a dozen times, ROS is easy, book keeping is what sucks.
- This is all you will need to know about the ROS filesystem for now, let's get to making your first node!
- First some terminology
  - Node - an executeable ROS program
  - Message - a data structure that can be passed over topics
  - Topic - a link between nodes that allows sharing of messages.
  - roscore - This *is* ROS. It contains the ROS master, rosout, and the ros parameter server.
  - roslaunch - an XML script to handle execution of multiple nodes and set parameters.

Node basics

- ROS only officially supports Python and C++. There have been attempts made at making Java and C implementations but they largely lack functionality.
- You can mix C++ and Python nodes without issue. As long as they use the right topics to talk to each other.
- Go ahead and Pick whichever language you would like to use for tonight's workshop. I primarily use python with ROS but I know how to use each so I'll be able to answer any questions.

Publishers & Subscribers

- In any ROS program there are two primary components. Publishers and Subscribers.
- A publisher is an object that interacts with a ROS topic to put data on the topic. For example a publisher might put some floating point data onto a topic called "velocity" to let any nodes know about a robot's speed.
- A subscriber is the counterpart to a publisher. A subscriber interacts with a ROS topic to take data off of it. Using the same example, after the publisher puts some velocity data on to the topic, some subscribers could then see that data and grab it for use in another Node.

Publisher program

- Open up your text editor of choice and if you havent already pick either C++ or Python. Doesn't matter if you know either language we can cover it.
- First lets handle getting ROS into our programs.
- C++:
  - #include "ros/ros.h"
  - #include "std_msgs/String.h"
  - #include <sstream.h>
- Python:
  - #!/usr/bin/env python
  - import rospy
  - from std_msgs.msg import String

Interacting with ROS

- C++:
  - Create a main function

```
int main(int argc, char** argv)
{
    ros::init(argc, argv, "talker");
    return 0;
}
```

- Python:

```
def talker():
    rospy.init_node('talker', anonymous=True)
```

Adding functionality

- C++:
  - ros::NodeHandle n;
  - ros::Publisher chatter_pub = n.advertise<std_msgs::String>("chatter", 1000);
  - ros::Rate loop_rate(10);
- Python:
  - pub = rospy.Publisher('chatter', String, queue_size=10)
  - rate = rospy.Rate(10)

Publishing Data

- C++:

```
int count = 0;
  while (ros::ok())
  {
    std_msgs::String msg;
    std::stringstream ss;

    ss << "hello world "<< count;
    msg.data = ss.str();
    ROS_INFO("%s",msg.data.c_str());

    chatter_pub.publish(msg);
    ros::spinOnce();
    loop_rate.sleep();
    ++count;
  }
```

- Python:

```
while not rospy.is_shutdown():
        hello_str = "hello world %s"       %
rospy.get_time()
        rospy.loginfo(hello_str)
        pub.publish(hello_str)
        rate.sleep()


if __name__ == '__main__':
    try:
        talker()
    except
rospy.ROSInterruptException:
        pass
```

Subscriber Program

- The subscriber program is pretty much the same as this publisher so there isn't much to gain by me going line by line with it.
- One major difference is worth mentioning.
  - Subscribers require a callback function that tell's the Node what to do with data received on a topic.
- Subscriber syntax:
  - C++:
    - ros::Subscriber sub = n.subscribe("topic_name", 1000, callback);
  - Python:
    - rospy.Subscriber("topic_name", String, callback)

Pros/Cons of ROS

- ROS allows you to make very simple modular programs.
- Can reduce the complexity of a robotics project that must do many tasks in tandem.
- Lots of great library support
- Book keeping is a nightmare
- Adds another level of abstraction
- Does nothing to help small projects

The End!

- This workshop only covered the fundamentals of ROS. Thers is so much more to learn.
- I have a few of my own program for ROS I am happy to provide anyone who wants to see how it has been used for real robotics applications.
- Experiment on you own, check out turtlesim and RVIZ.
- We have ROS enabled robots and sensors in the LAIR too!
    - Turtlebot
    - Corobot
    - Roadrunner
    - iCreate Base
    - Kinect