BAHRIA UNIVERSITY ISLAMABAD

ROBOTICS LAB

LAB 2: INTRODUCTION TO ROBOT OPERATING SYSTEM - 1

Name of Student: …………………………………………………….

Roll No.: ………………………………………………………………….

Date of Experiment: …………………………………………………

Report submitted on: ……………………………………………….

Marks obtained: ……………………………………

Remarks: ……………………………………………

Instructor's Signature: ……………………………..

Fall 2021

Goals

By the end of this lab, you should be able to:

• Set up a new ROS environment, including creating a new workspace and creating a package with the appropriate dependencies specified.

• Use the catkin tool to build the packages contained in a ROS workspace Run nodes using rosrun.

• Use ROS's built-in tools to examine the topics and services used by a given node.

What is ROS ?

ROS is an open-source, meta-operating system for your robot. It provides the services you would expect from an operating system, including hardware abstraction, low-level device control, implementation of commonly-used functionality, message-passing between processes, and package management. It also provides tools and libraries for obtaining, building, writing, and running code across multiple computers.

In ROS, all the major functionality is broken up into a number of chunks that communicate with each other using messages. Each chunk is called a node and is typically run as a separate process. Matchmaking between nodes is done by the ROS master.

ROS Master

The ROS Master provides naming and registration services to the rest of the nodes in the ROS system. It tracks publishers and subscribers to topics as well as services. The role of the Master is to enable individual ROS nodes to locate one another. Once these nodes have located each other they communicate with each other peer-to-peer.

he Master is most commonly run using the roscore command, which loads the ROS Master along with other essential components.

Understanding ROS Nodes

A node is simply an executable file that performs some tasks. Nodes exchange control messages, sensor readings, and other data by publishing or subscribing to topics or by sending requests to services offered by other nodes (these concepts will be discussed in detail later in the lab). Nodes can be written in a variety of languages, including Python and C++, and ROS transparently handles the details of converting between different datatypes, exchanging messages between nodes, etc.

ROS nodes use a ROS client library to communicate with other nodes. ROS client libraries allow nodes written in different programming languages to communicate:

- rospy = python client library
- roscpp = c++ client library

To run a rosnode you first have to run the ros master and then run the command for rosnode. Rosrun is used to run a node and you have write package name and node name that you want to run in this command.

- $ rosrun [package_name] [node_name]

Once the nodes start running, you can use command “rosnode list” to keep track on what nodes are currently running.

- $ rosnode list.

ROS Topic

Topics are named buses over which nodes exchange messages. Topics have anonymous publisher/subscriber semantics. A node does not care which node published the data it receives or which one subscribes to the data it publishes. There can be multiple publishers and subscribers to a topic.

Rostopic contains the rostopic command-line tool for displaying debug information about ROS Topics, including publishers, subscribers, publishing rate, and ROS Messages. It also contains an experimental Python library for getting information about and interacting with topics dynamically. The rostopic command-line tool displays information about ROS topics. Currently, it can display a list of active topics, the publishers and subscribers of a specific topic, the publishing rate of a topic, the bandwidth of a topic, and messages published to a topic. The display of messages is configurable to output in a plotting-friendly format.

```
rostopic bw     display bandwidth used by topic
rostopic delay display delay for topic which has header
rostopic echo   print messages to screen
rostopic find   find topics by type
rostopic hz     display publishing rate of topic
rostopic info   print information about active topic
rostopic list   print information about active topics
rostopic pub    publish data to topic
rostopic type   print topic type
```

ROS Messages

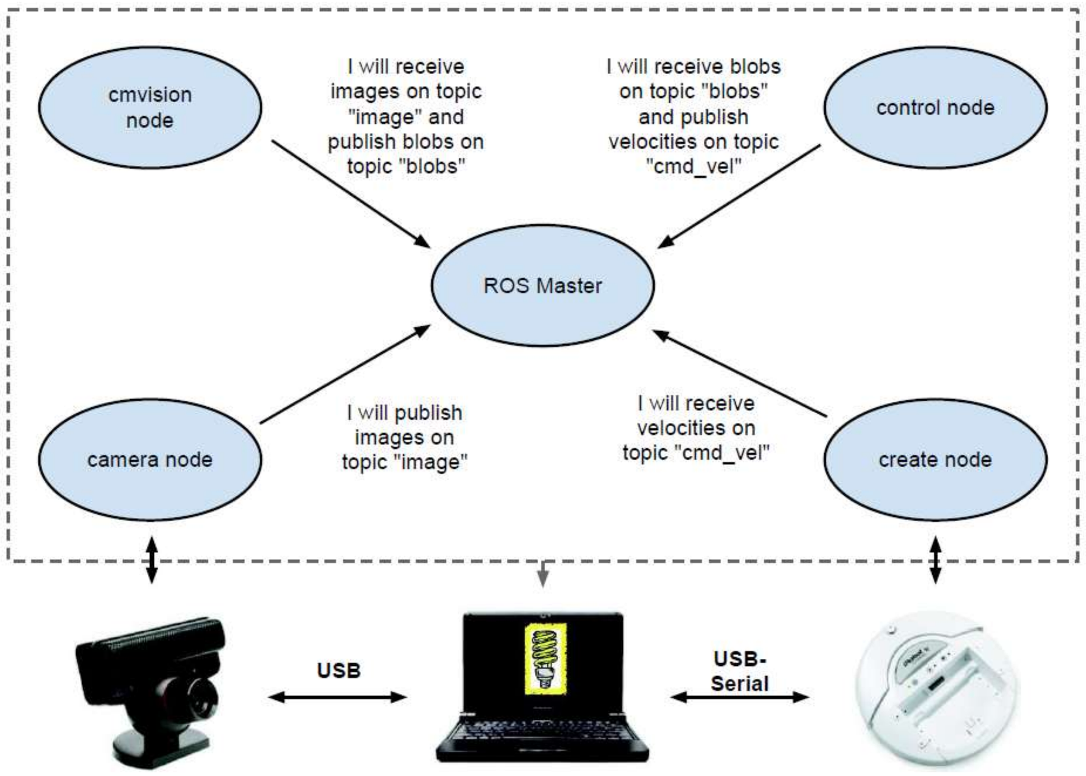
Nodes communicate with each other by publishing messages to topics. A message is a simple data structure, comprising typed fields. Standard primitive types (integer, floating point, boolean, etc.) are supported, as are arrays of primitive types. Messages can include arbitrarily nested structures and arrays (much like C structs).

Some types of messages are as fellows:

- Std_msgs/bool
- Std_msgs/int32
- Std_msgs/string
- Std_msgs/float64

You can study more about the types of ros messages on this link .

ROS Overview


cmvision node
I will receive images on topic "image" and publish blobs on topic "blobs"
I will receive blobs on topic "blobs" and publish velocities on topic "cmd_vel"
control node
ROS Master
camera node
I will publish images on topic "image"
I will receive velocities on topic "cmd_vel"
create node
USB
USB-Serial

[adapted from slide by Chad Jenkins]

For instance, let's say we have four Nodes; a Camera node and an cmvision node. A typical sequence of events would start with Camera notifying the master that it wants to publish images on the topic "images":

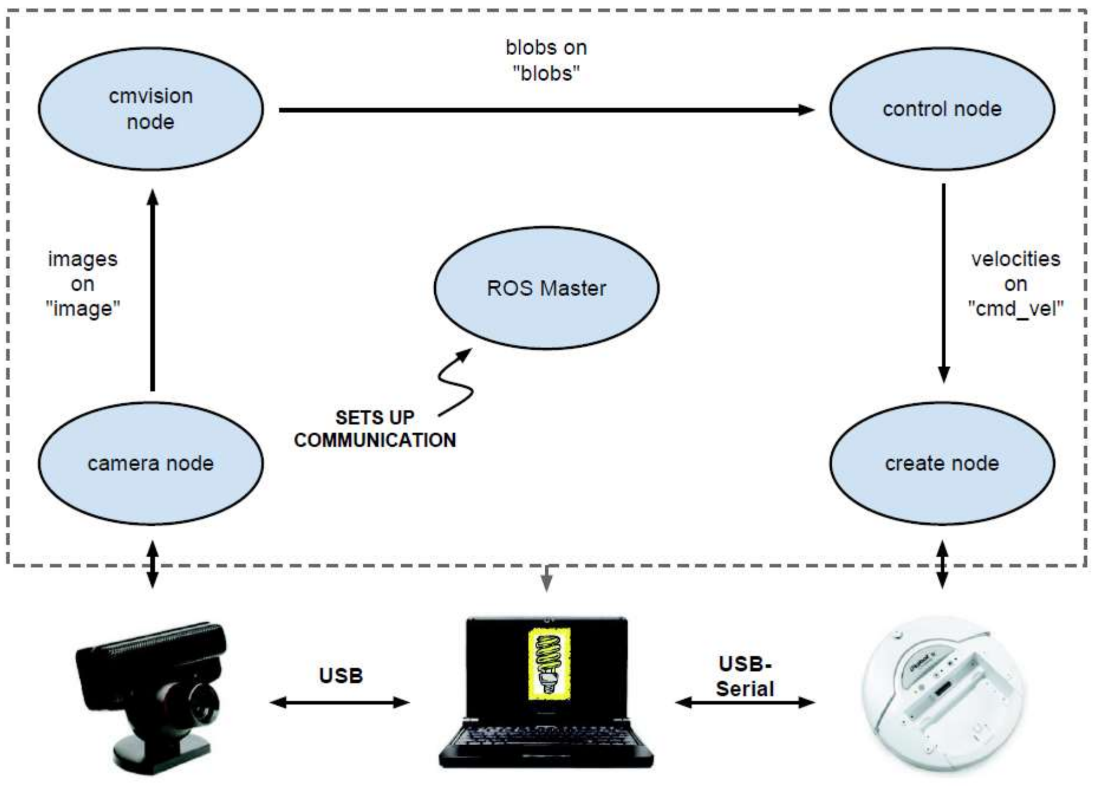
Now, Camera publishes images to the "images" topic, but nobody is subscribing to that topic yet so no data is actually sent. Now, Image_viewer wants to subscribe to the topic "images" to see if there's maybe some images there. cmvision node tells the rosmaster that it wants to subscribe the topic images and publish on topic blobs. Once camera node start publishing images, cmvision node starts receive these images.

Control node tells the ros master that it wants to subscribe the topic blobs and wants to publish on topic velocities. As soon as cmvision start publishing blobs, control node will receive the blobs and compute velocities.

Create node tells the rosmaster that it wants to subscribe the topic velocities.

Now that the topic "images", "blobs" and "velocity" has both a publisher and a subscriber, the master node notifies Camera, cmvision, control, create node about each other's existence so that they can start transferring messages to one another:

That's how the figure becomes, once the rosmaster sets up the communication.



[adapted from slide by Chad Jenkins]

ROS Installation

Setup your computer to accept software from packages.ros.org.

```
sudo sh -c 'echo "deb http://packages.ros.org/ros/ubuntu $(lsb_release -sc) m
ain" > /etc/apt/sources.list.d/ros-latest.list'
```

Setting up the keys

```
sudo apt install curl # if you haven't already installed curl

curl -s https://raw.githubusercontent.com/ros/rosdistro/master/ros.asc | sudo
apt-key add -
```

Before moving to the Installation, make sure your Debian package index is up-to-date:

```
sudo apt update
```

After this, run this command to install full desktop version of ROS.

```
sudo apt install ros-noetic-desktop-full
```

You must source this script in every bash terminal you use ROS in

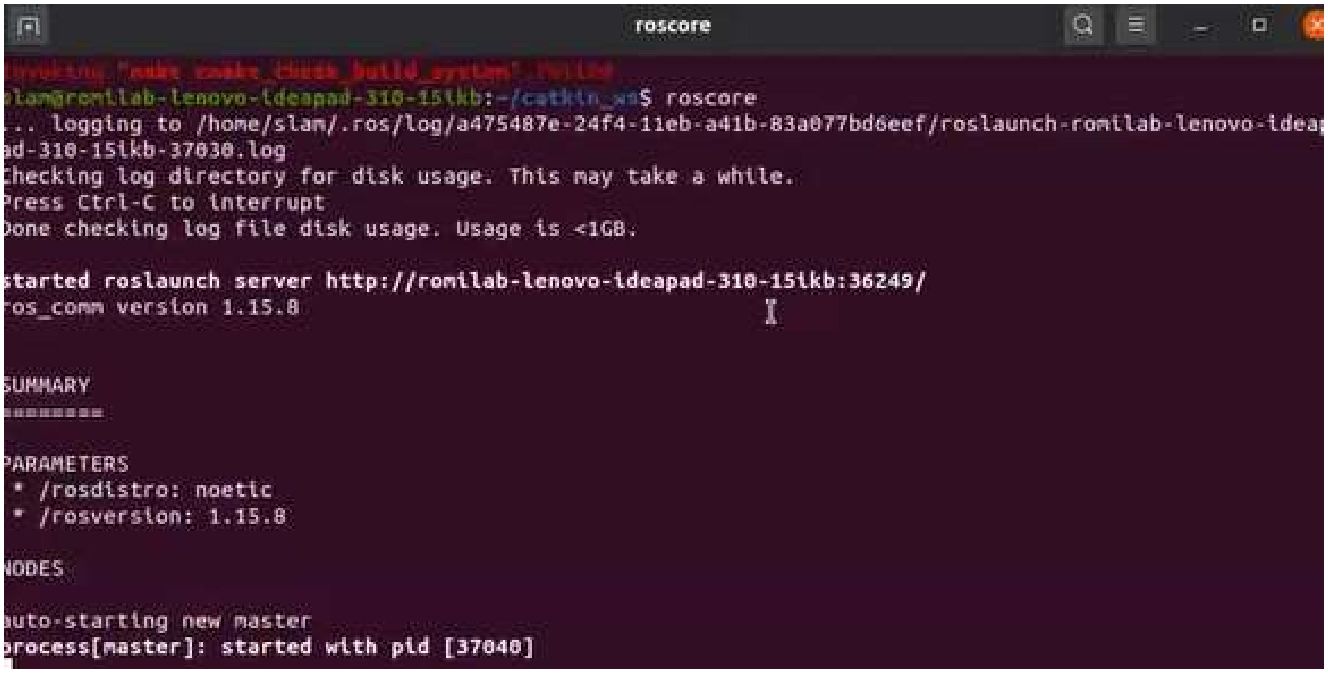
```
source /opt/ros/noetic/setup.bash
```

It can be convenient to automatically source this script every time a new shell is launched. These commands will do that for you.

```
echo "source /opt/ros/noetic/setup.bash" >> ~/.bashrc

source ~/.bashrc
```

After successful installation if you run commad roscore it should give output like this:



```
roscore
nvoking "make cmake_check_build_system" failed
lam@romilab-lenovo-ideapad-310-15ikb:~/catkin_ws$ roscore
.. logging to /home/slam/.ros/log/a475487e-24f4-11eb-a41b-83a077bd6eef/roslaunch-romilab-lenovo-idea
ad-310-15ikb-37030.log
hecking log directory for disk usage. This may take a while.
ress Ctrl-C to interrupt
one checking log file disk usage. Usage is <1GB.

tarted roslaunch server http://romilab-lenovo-ideapad-310-15ikb:36249/
os_comm version 1.15.8


UMMARY
=======

ARAMETERS
 * /rosdistro: noetic
 * /rosversion: 1.15.8

ODES

uto-starting new master
rocess[master]: started with pid [37040]
```

Creating a ROS Workspace

This tutorial assumes that you have installed catkin and sourced your environment. If you installed catkin via apt-get for ROS noetic, your command would look like this:

```
$ source /opt/ros/noetic/setup.bash
```

Let's create and build a catkin workspace:

```
$ mkdir -p ~/catkin_ws/src

$ cd ~/catkin_ws/

$ catkin_make
```

The catkin_make command is a convenience tool for working with catkin workspaces. Running it the first time in your workspace, it will create a CMakeLists.txt link in your 'src' folder.

Additionally, if you look in your current directory you should now have a 'build' and 'devel' folder. Inside the 'devel' folder you can see that there are now several setup.*sh files. Before continuing source your new setup.*sh file:

```
$ source devel/setup.bash
```

To make sure your workspace is properly overlayed by the setup script, make sure ROS_PACKAGE_PATH environment variable includes the directory you're in.

```
$ echo $ROS_PACKAGE_PATH

/home/youruser/catkin_ws/src:/opt/ros/kinetic/share
```

Creating a Ros Packege

First change to the source space directory of the catkin workspace you created in the Creating a Workspace for catkin tutorial:

```
# You should have created this in the Creating a Workspace Tutorial

$ cd ~/catkin_ws/src
```

Now use the catkin_create_pkg script to create a new package called 'lab1s' which depends on std_msgs, roscpp, and rospy:

```
 catkin_create_pkg lab1 std_msgs rospy roscpp
```

This will create a lab1 folder which contains a package.xml and a CMakeLists.txt, which have been partially filled out with the information you gave catkin_create_pkg.

catkin_create_pkg requires that you give it a package_name and optionally a list of dependencies on which that package depends.

Now you need to build the packages in the catkin workspace:

```
$ cd ~/catkin_ws

$ catkin_make
```

After the workspace has been built it has created a similar structure in the devel subfolder as you usually find under /opt/ros/$ROSDISTRO_NAME.

To add the workspace to your ROS environment you need to source the generated setup file:

```
$ . ~/catkin_ws/devel/setup.bash
```

Conclusion

After successful implementation of this lab, your working directory should look like this.

```
->ros_workspaces

   ->build

   ->devel

       -- setup.bash

   ->src

       --CMakeLists.txt

       ->Lab1

           -- CMakeLists.txt

           -- package.xml

           -- package.xml

           -- include
```

Tasks

- Install the ROS and run command “roscore” to check if the ros is correctly installed.
- Create a Ros workspace and run command “echo $ROS_PACKAGE_PATH” to check if workspace is properly overlayed by the setup script.
- Create a package by your group name and Explain the contents of your ~/ros_workspaces directory.
- Demonstrate the use of the catkin_make command.
- Use ROS’s utility functions to get data about packages.