Lab02 – Jetson Nano and ROS

ELEC ENG 3EY4

Brent Menheere - menheerb – 400362843

Aum Shah – shaha124 – 400388075

L04

February 1, 2024

Question 1:

When creating a ROS package we need the following dependencies:

`std_msgs`: This dependency is used to provide standard message types, which include strings, floats, and ints. When publishing messages the data is in the form of one of these message types.

`roscpp`: This dependency is used to provide the C++ library for ROS, this allows the ROS system to manage and create node in C++.

`rospy`: This dependency is similar to the last, however will Python instead.

Question 2:

`sudo`: This command stands for super user do and lets the user act with privileged access to the system and run restricted commands.

`apt-get`: The Advanced Packaging Tool application which allows us to install, remove, and upgrade packages.

`install`: Can be used with apt-get to install packages.

`ros-melodic-serial`: Allows for sending of messages over serial communication. In the case of our system we use RS232 as means of serial communication.

`ros-melodic-ackermann-msgs`: This command relates to the Ackermann steering control package.

`ros-melodic-rplidar`: This command relates to the package that supports the LiDAR on our car.

`ros-melodic-realsense2-camera`: This package provides nodes for the use of the camera on board our RC car.

`libusb-dev`: This package allows for the device to communicate with devices via USB.

`libspnav-dev`: Provides libraries used for interfacing with 3D devices.

Question 3:

Based on what we learned in Lab 1, the command we used to delete the Wiimote driver was `sudo rm -rm wiimote`. Before we used this command, we first needed to navigate to the joystick drivers directory which had the path `~/catkin_ws/src/joystick_drivers` then the above command could be implemented. The method of using `-r` is similar to Lab 1 when deleting a directory, we needed to recursively use the `rm` command to delete everything inside of the directory including itself.

Question 4:

To build all the packages we use the command `catkin_make`. This command builds the ROS catkin workspace. We used this to build the packages and ensure that our ROS program is ready for execution. The build process compiles the source code and generates other necessities needed to operate ROS nodes and application.

Question 5:

We modified the ".bashrc" file to add setup.bash as a source. This is required to configure the environment variables needed for ROS to work as intended. It will make sure that the ROS workspace is recognised as a source, and that the ROS tools can find and utilize the packages we have installed in our catkin workspace.

Question 6:

In Figure 20, we used the commands `/rosout` and `/rosout_agg`. The rosout topic is used as a console log reporting mechanism. Each log from a node is published in the rosout topic. The rosout_agg topic "aggregates" (groups together) log messages from multiple nodes into a message feed that can be received directly. This is useful since you can monitor log messages in one place.

Question 7:

The command used in Figure 21 was: `rostopic pub /help std_msgs/String "Hello Robot"`

`rostopic pub`: This command is used to publish data to a topic.

`/help`: This command specifies which ROS topic the message will be published to.

`std_msgs/String`: This command declares the type of message which is being published. In this case we want to send a message as a string since we are sending a string of characters.

`"Hello Robot"`: This is the message that is being sent.

Overall, we can analyze the line of code to deduce that to publish data to a topic we first use the `rostopic pub` command, then declare where we want to send the data, declare the data type, and finally define what our data/message is.

Question 8:

```
brentandaum@brentandaum-desktop:~$ rosnode info /rostopic_12106_1706208407405
--------------------------------------------------------------------------------
Node [/rostopic_12106_1706208407405]
Publications:
 * /hello [std_msgs/String]

Subscriptions: None

Services:
 * /rostopic_12106_1706208407405/get_loggers
 * /rostopic_12106_1706208407405/set_logger_level


contacting node http://brentandaum-desktop:38693/ ...
Pid: 12106
```

Figure 1 Output of rosnode info /rostopic...

```
brentandaum@brentandaum-desktop:~$ rostopic info /hello
Type: std_msgs/String

Publishers:
 * /rostopic_12106_1706208407405 (http://brentandaum-desktop:38693/)

Subscribers: None
```

Figure 2 Output of rostopic info /hello

In this task, we initiated the Robot Operating System (ROS) by launching `roscore`. Subsequently, we established a topic and published the string "Hello Robot" using `rostopic pub /hello std_msgs/String "Hello Robot."` Following this, we displayed the data from the topic on the terminal using `rostopic echo /hello`. Identifying the node responsible for publishing the topic was accomplished through the utilization of `rosnode list`. Once the node was identified, we

employed `rosnode info /rostopic…` to retrieve information about the node. An alternative to this step would be using `rostopic info /hello`.

To summarize, we gained insights into ROS nodes and topics, becoming acquainted with a range of commands for initializing ROS, creating topics, displaying data from topics, and listing nodes.