**Exploring ROS2 nodes and topics using the MentorPi robot.**

Important note:

* The tab key gives the ability to autocomplete commands to prevent entering the entire line.

1. Stop app control with the following command.

**~/.stop\_ros.sh**

1. Enter the command below.

**ros2 node list**

Notice that nothing appears since there are no nodes that are on the ROS2 network.

1. Enter: **ros2 topic list**

You should see two topics shown. These are internal to ROS2 itself and are used to manage the specific nodes and topics.

1. Enter the following:

**ros2 launch ros\_robot\_controller ros\_robot\_controller.launch.py**

This command starts a launch file that can be configured to do a variety of things related to the robots operation. In particular, this launch file starts the **ros\_robot\_controller** node as well as handles a few other configuration tasks.

Confirm that the node mentioned above is running by entering: **ros2 node list**

Check which topics the node establishes with the command: **ros2 topic list**

1. Let’s look at HOW messages are sent on specific topics. We will chose the /ros\_robot\_controller/set\_rgb topic and look at what **interface** it uses.

Enter: **ros2 topic info /ros\_robot\_controller/set\_rgb**

You should see that this topic has one subscriber and no publishers. That means that one other node is listening to messages that this topic broadcasts.

You should see under ‘type’ that there is an interface for this topic that is called ‘RGBStates’

Enter: **ros2 interface show ros\_robot\_controller\_msgs/msg/RGBStates**

You will see that there are four data types that this interface uses:

int32 index

uint8 red

uint8 green

uint8 blue

This represents which led to trigger and the value it should take

You are now ready to send commands (publish) commands on this topic using the interface queried.

NOTE: Some topics such as set\_motor require other nodes to be launched. SO while we can do the same for this (and any other) topic, we might not be able to do anything useful yet.

1. Let’s publish to the topic to remontely trigger an LED on the robot enter the following (Don’t press enter until everything is typed it as shown):

**ros2 topic pub /ros\_robot\_controller/set\_rgb ros\_robot\_controller\_msgs/msg/RGBStates "{states: [**

**{index: 1, red: 255, green: 0, blue: 0},**

**{index: 2, red: 255, green: 0, blue: 0}**

**]}"**

Notice the text on your screen that says publishing #<n> and that the two LEDs on your robot have turned red.  
This confirms that your message is being received by a **node that subscribes to the /ros\_robot\_controller/set\_rgb topic**.  
That node runs code which processes the message and **physically sets the LED colors** on the robot.

1. Try it – use what you learned to learn about the /ros\_robot\_controller/battery topic.

What is the msg type (interface) used by the battery topic: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Notice that /ros\_robot\_controller/battery is a **publisher** topic — this means a node is **sending data** on this topic, and any other node can subscribe to receive the battery readings. This topic **does not receive** data; it only **publishes** battery status updates.

9) We can view the data that is being published by using: **ros2 topic echo /ros\_robot\_controller/battery**

We can observe something like this: data: 7651

This represents battery voltage in mV, so 7651mV = 7.65V

Now let’s start digging into the system that allows the robot to move efficiently by calculating its position. There are two types of data that need to be considered, IMU and odometry.

IMU calculates the acceleration and angular velocity of the robot in each of its three axes.

Odometry uses encoders on the wheels to calculate the motion of the wheels.

1. We can start the main controller as in step 4:

**ros2 launch ros\_robot\_controller ros\_robot\_controller.launch.py**

1. We then need to start the IMU controller:

**ros2 launch peripherals imu\_filter.launch.py**

1. If we examine the new topics that the IMU introduced: **ros2 topic list**

…. We can see that there is a new /imu and /imu\_corrected topic.

By typing **ros2 topic info /imu**, we can see that one node is publishing to this topic — which means that IMU data is being sent and is available for other nodes to receive.

1. A computer screen with white text

   AI-generated content may be incorrect.Just like with the battery topic, we can view the data being published on the /imu topic using:

**ros2 topic echo /imu**

There is a lot of data that moves fast. It should look something like the image here:

You might notice that if you move the robot some of the numbers will begin to change. This shows a change in the **angular velocity** and **linear acceleration**

1. Now let’s view the odometry data. Start your controller (or ensure it is running):

**ros2 launch ros\_robot\_controller ros\_robot\_controller.launch.py**

1. Look for the **/odom** topic in your topic list

**ros2 topic list**

We can also see HOW MANY topic are running (we should see 33):

**ros2 topic list | wc -l**

1. Let’s view data that shows the position and orientation of the robot in the three axes.

**ros2 topic echo /odom**

While the topic data is displayed move the robot around. While moving the robot press **ctrl+c**. Then scroll up to observe the data. This makes it a little easier to see changes then to keep up with the data being echoed.

Another useful source of data that shows both linear acceleration and ngular velocity is: **/imu\_corrected**

Let’s now drive the robot by publishing data on linear and angular velocity.

The **linear velocity** (relative to the robot) is the x-axis which represents forward and backward movement. This can range from about **-0.6 to 0.6 (m/sec)**

The **angular velocity** (relative to the robot) is z-axis which represents the left and right turn. This can range from **-2 (full right) to 2 (full left)**

1. The /cmd\_vel command is what enables high level movement control (common in many ROS applications). Let’s see what type of message the cmd\_vel topic uses:

**ros2 topic info /cmd\_vel**

We see - **Type:** **geometry\_msgs/msg/Twist**

18) Let’s inspect how these messages are structured:

**ros2 interface show geometry\_msgs/msg/Twist**

This shows that the message contains:

* A linear field with 3 float values (x, y, z)
* An angular field with 3 float values (x, y, z)  
  These are used to represent linear and angular velocity in 3D space.

19) Let’s send move commands by publishing to the velocity control topic using the message type we observed:

**ros2 topic pub /controller/cmd\_vel geometry\_msgs/Twist "linear:**

**x: 0.2**

**y: 0.0**

**z: 0.0**

**angular:**

**x: 0.0**

**y: 0.0**

**z: 0.0"**

This command tells the robot to **drive forward at 0.2 m/s with no rotation**.

⚠️ Make sure the motion controller node is running (ros2 launch controller controller.launch.py), or the robot will not respond.

⚠️ Make sure the robot is on a flat surface and has room to move

⚠️ It’s a good idea to have a second window open and ready to send a follow up stop command whenever a move command of this sort is published.

**ros2 topic pub /controller/cmd\_vel geometry\_msgs/Twist "linear:**

**x: 0.0**

**y: 0.0**

**z: 0.0**

**angular:**

**x: 0.0**

**y: 0.0**

**z: 0.0"**