***Introduction to Robotic Systems Course***

**LAB 06a**

**ROS Computation Graph: Communication between Nodes**

**Issue 1.0**

Contents

[1 Introduction 1](#_Toc30064225)

[1.1 Lab Overview 1](#_Toc30064226)

[2 Requirements 1](#_Toc30064227)

[3 ROS Computation Graph 1](#_Toc30064228)

[4 Task: Writing a Topic Publisher 2](#_Toc30064229)

[4.1 Ensure Topic Publisher is Connected 2](#_Toc30064230)

[4.2 Write Topic Publisher Program in Arduino 2](#_Toc30064231)

[5 Task: Display Message in Serial Monitor 3](#_Toc30064232)

[5.1 Start Rosserial 3](#_Toc30064233)

[6 Task: Visualize the Graph 4](#_Toc30064234)

# Introduction

## Lab Overview

This lab introduces some basic ROS Computation Graph concepts that enable communication between nodes. These concepts include nodes, topics, services.

In this lab, we will enable communication between the microcontroller and the remote computer using the Rosserial package for serial transmission. This will enable us to display the readings from the four IR sensors on a serial monitor.

# Requirements

The following hardware and software are required to complete this lab:

* **Hardware:** 
  + TurtleBot 3 Burger.
  + Four TCRT5000 IR reflector sensors (shown in Figure 1).
* **Software:** Arduino IDE.

# ROS Computation Graph

In ROS, a node is the minimum unit of an executable program that performs a computation and uses the ROS client library to communicate with other nodes.

The ROS computation graph is a distributed network of ROS nodes that partition tasks and exchange data.

A node can supply or consume message to a topic, in which case, we say the node is a publisher or subscriber, respectively.

ROS computation graph has several communication methods that are topics and services. We use a topic when the communication needs to be unidirectional, asynchronous, and continuous.

The publisher node broadcasts a message on the topic, while the subscriber node receives the message from the topic. For example, a node can publish a message containing the readings of the Laser Distance Sensor on a topic while several other nodes can subscribe to that topic to access the message and thus use them for some tasks.

A service allows bidirectional synchronous communication in a request and response communication between nodes. The nodes can be in a one-to-one communication that, when completed, causes the termination of the connection.

# Task: Writing a Topic Publisher

In this task, we will write a simple program that allows a publisher node to publish the IR sensor readings to a topic. We will connect the topic publisher to the OpenCR1.0 board, which will also run the code that we will write. Our topic publisher will be the four IR sensors.

## Ensure Topic Publisher is Connected

Connect the four IR sensors to the OpenCR1.0 board analogue input pins as shown in the line sensing lab.

## Write Topic Publisher Program in Arduino

Open Arduino IDE on the Remote computer and create a new project. On the sketch, enter the following lines of code to include ROS and standard message library header files.

1. #include <ros.h>
2. #include <std\_msgs/Byte.h>

Define the analogue pins for the IR sensors using the lines of code below.

1. #define sensor\_LL A2
2. #define sensor\_ML A0
3. #define sensor\_MR A1
4. #define sensor\_RR A3

Next, declare a node handler named “nh,” a byte message named “IR\_msg” and instantiate an object of the topic publisher class. Here, the topic name is “IR.” These steps are shown in the three lines of code below.

1. ros::NodeHandle nh;
2. std\_msgs::Byte IR\_msg;
3. ros::Publisher pub\_ir("IR", &IR\_msg);

In the setup function, call the **initNode()** function. This creates a node. Next, advertise the Publisher **pub\_ir** to enable the node publish data.

1. void setup()
2. {
3. nh.initNode();
4. nh.advertise(pub\_ir);
5. }

In the loop function, set a time interval of 50 ms to publish the data readings from the IR sensors 20 times per second. The **spinOnce()** function makes sure any call-back functions are called in the subscriber nodes (if any).

1. void loop()
2. {
3. uint8\_t reading = 0;
4. static uint32\_t pre\_time;
5. reading = (analogRead(sensor\_LL) > 700) <<3|(analogRead(sensor\_ML) > 700) << 2 | (analogRead(sensor\_MR) > 700)<<1|(analogRead(sensor\_RR) > 700)<<0;
6. if (millis()-pre\_time >= 50)
7. {
8. IR\_msg.data = reading;
9. pub\_ir.publish(&IR\_msg);
10. pre\_time = millis();
11. }
12. nh.spinOnce();
13. }

Note, since the outputs from the IR sensors are digital signals, which drive analogue inputs on the OpenCR1.0 board, we set a threshold of 700 as a trigger point for the detected lines. The result is “1” if the IR reading is greater than 700, which is a good threshold to detect the black line and “0” if below 700. With this, we can encode the signals from the IR sensors as a 4-bit number, which has a range of 0 to 15.

Click the **Verify** and **Upload** buttons to compile and upload the code to the OpenCR1.0 board. The code should run upon reset.

# Task: Display Message in Serial Monitor

For this task, we will output the topic message (IR sensors reading) to a serial monitor. First, we will connect the microcontroller to the PC using Rosserial package for serial transmission.

## Start Rosserial

Rosserial is a package that enables serial transmission (e.g., UART) of ROS messages between the PC and microcontrollers.

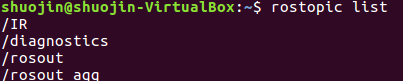
Open a terminal in the Remote PC Ubuntu and enter the following command.

1. $ roscore
2. $ rosrun rosserial\_python serial\_node.py \_\_name:=opencr \_port:=/dev/ttyACM0 \_baud:=115200

Run this command to see if the IR topic is being published:

$ rostopic list

If everything works fine, you would see a similar output to the one shown in Figure 1.

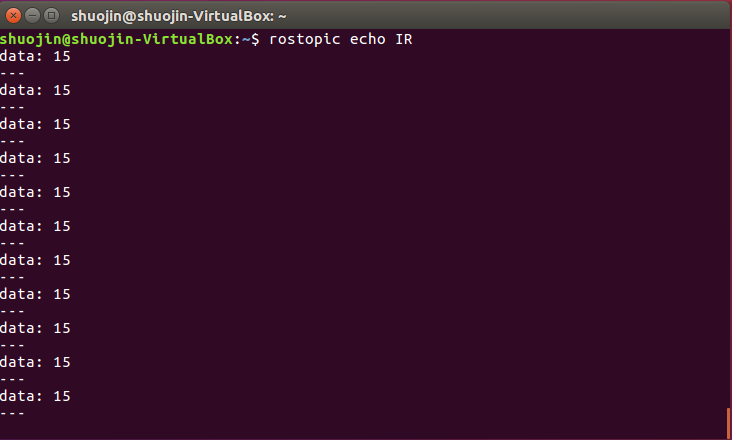


*Figure 1: ROS topic list.*

Finally, check the data and run the following command to do this:

$ rostopic echo IR

The IR data will be returned as an integer number with values from 0 to 15 depending on the IR sensor’s detection. See Figure 2 below.



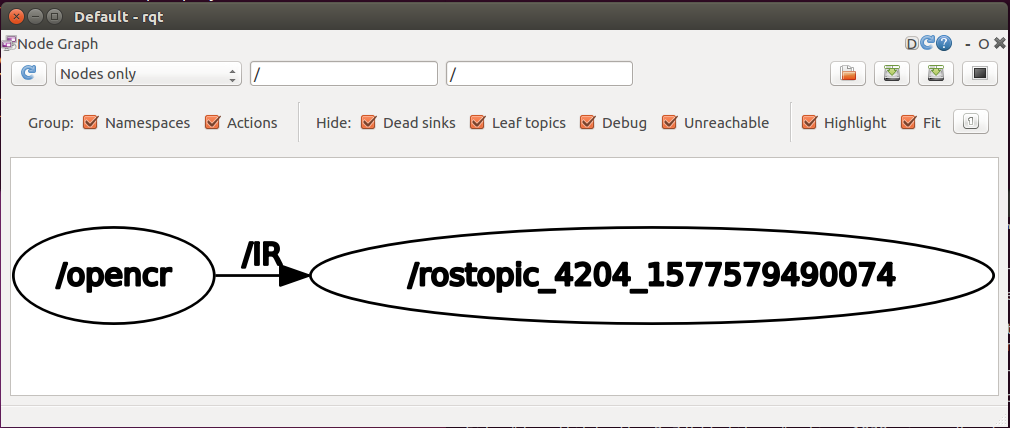
*Figure 2: Serial output of IR sensor.*

# Task: Visualize the Graph

Open “rqt” to visualize the node diagram. And we can see the IR message is successfully published.

$ rqt

If a blank rqt window opens, hover your mouse at the top of your window (where the title “Default – rqt” is) and click **Plugging** -> **Introspection** -> **Node Graph**. This will display the node diagram of the ROS Graph as shown in Figure 3.



*Figure 3: Node diagram of the ROS Graph.*