7

Interfacing I/O Boards, Sensors, and Actuators to ROS

* Understanding the Arduino-ROS interface

⇒ Arduino boards are powered by Atmel microcontrollers, which are available from 8-bit to 32-bit and clock speed from 8 MHz to 84 MHz.

⇒ Arduino ⇒ Used for quick prototyping of robots.

⇒ Main Application

→ Interfacing sensors and actuators and communicating with PC

- Arduino UNO → Basic robotics & Sensor Interfacing
- Arduino Mega 2560 → Intermediate robotics application level application
- Arduino Due → High end robotics Application

⇒ Most of the Communication between PC and I/O board in robots will be through UART protocol.

→ We can implement our own logic to receive and transmit the data from board to PC and Vice Versa

⇒ The Arduino-ROS interface is a standard way of communication between the Arduino boards and PC.

⇒ We can use the similar C++ APIs of ROS used in PC in Arduino IDE also, for programming the Arduino board.

★ Understanding the rosserial package in ROS

→ Set of standardized communication protocols implemented for communicating from ROS to Character devices such as serial ports and sockets and Vice Versa

⇒ The rosserial protocol can convent the standard ROS messages and service data types to embedded device equivalent data type.

⇒ The Serial data is sent as data Packets by adding header and tail bytes on the packet.

1st Byte	Sync Flag (Value: 0xff)

2nd Byte	Sync Flag / Protocol version

⇒ This byte was 0xff on ROS Groovy and after that it is set to 0xfe

3rd Byte	Message Length (N) – Low Byte

4th Byte	Message Length (N) – High Byte

⇒ This is the length of the packet

5th Byte	Check Sum over message length

⇒ For finding packet Corruption

6th Byte 7th Byte	Topic ID

N Byte	Serialized Message Data

⇒ This is the data associated with each topic.

(N + 8) Bytes	Checksum over Topic ID & Message Data

⇒ Send data for finding packet Corruption.

⇒ The Checksum of length is computed using the following equation:

$$\text{Checksum} = 255 - \left(\left(\text{Topic ID Low Byte} + \text{Topic ID High Byte} + \ldots \text{data byte value} \right) \% 256 \right)$$

⇒ rosserial_client library helps us to develop ROS nodes from :-

1) Arduino ⇒ rosserial_arduino

2) Embedded Linux Platform ⇒ rosserial_embeddedlinux

3) Windows ⇒ rosserial_windows.

⇒ On the PC side, we need some other Packages to decode the serial messages & Convert to exact topic from the ros_serial_client libraries.

1) rosserial_python
   → The receiving node is completely written in Python

2) rosserial_server
   → Inbuilt functionalities are less Compared to rosserial_python, but it can be used for high Performance application.

★ Installing rosserial packages on Ubuntu

1. Install the rosserial package binaries using apt-get.

```
sudo apt-get install ros-melodic-rosserial
ros-melodic-rosserial arduino
ros-melodic-rosserial-server
```

2. Install latest Arduino IDE
3. Install JAVA runtime support if it is not installed.

```
sudo apt-get install java-common
```

4. Set the location of your sketches in Arduino IDE Preferences.

5. Create a folder name ~~Ros_lib~~ libraries in the Sketchbook location.

6. Now we can generate ros_lib using a script called make_libraries.py, which is present inside the rosserial_arduino package.

rosrun rosserial_arduino make_libraries.py <Path the library folder>

★ Understanding ROS node APIs in Arduino

⇒ Basic structure of ROS Arduino node.

```
#include <ros.h>
ros::NodeHandle nh;
Void setup()
{
 nh.initNode();
}
Void loop()
{
 nh.spinOnce();
}
```

⇒ We can create the Subscriber and Publisher object in Arduino, similar to the other ROS client libraries.

Defining Subscriber object in Arduino:-

```
ros::Subscriber<std_msgs::String>
    sub("talker", callback);
```

↓

We define a Subscriber which is subscribing a String message, where the Callback is the callback function executing when a string message arrives on the talker topic

Defining Publisher object in Arduino

```
ros::Publisher chatter("chatter", &str_msg);
    chatter.publish(&str_msg);
```

⇒ After defining the publisher and the Subscriber, we have to initiate this inside the setup() function using the following lines of code.

```
nh.advertise(chatter);
nh.subscribe(sub);
```

⇒ There are ROS APIs for logging from Arduino.

```
nh.logdebug("Debug Statement")
nh.loginfo("Program info")
nh.logwarn("Warnings")
nh.logerror("Error")
nh.logfatal("Fatalities!")
```

⇒ We can retrieve the current ROS time in Arduino using ROS built in functions such as time and duration.

- Current ROS time

```
ros::Time begin = nh.now();
```

- Convert ROS time in seconds:

```
double secs = nh.now().toSec();
```

- Creating a duration in seconds:

```
ros::Duration ten_seconds(10, 0);
```

* Interfacing Non-Arduino board to ROS

⇒ In such case, we may want to write our own drivers for the board which can convert the serial messages into topics.