Orientation sensor documentation
Adafruit BNO055

Author: Ban Bogdan

Contents

Introduction

The purpose of this project is to extract data from the BNO055 9-axis absolute orientation sensor. The device works as an accelerometer, gyroscope and magnetometer and can collect other useful information from the environment, like temperature, Euler heading, quaternion etc. (official documentation: BST_BNO055_DS000_12 (adafruit.com))

Diagram of classes

The UML diagram of the project is presented in Figure 1:

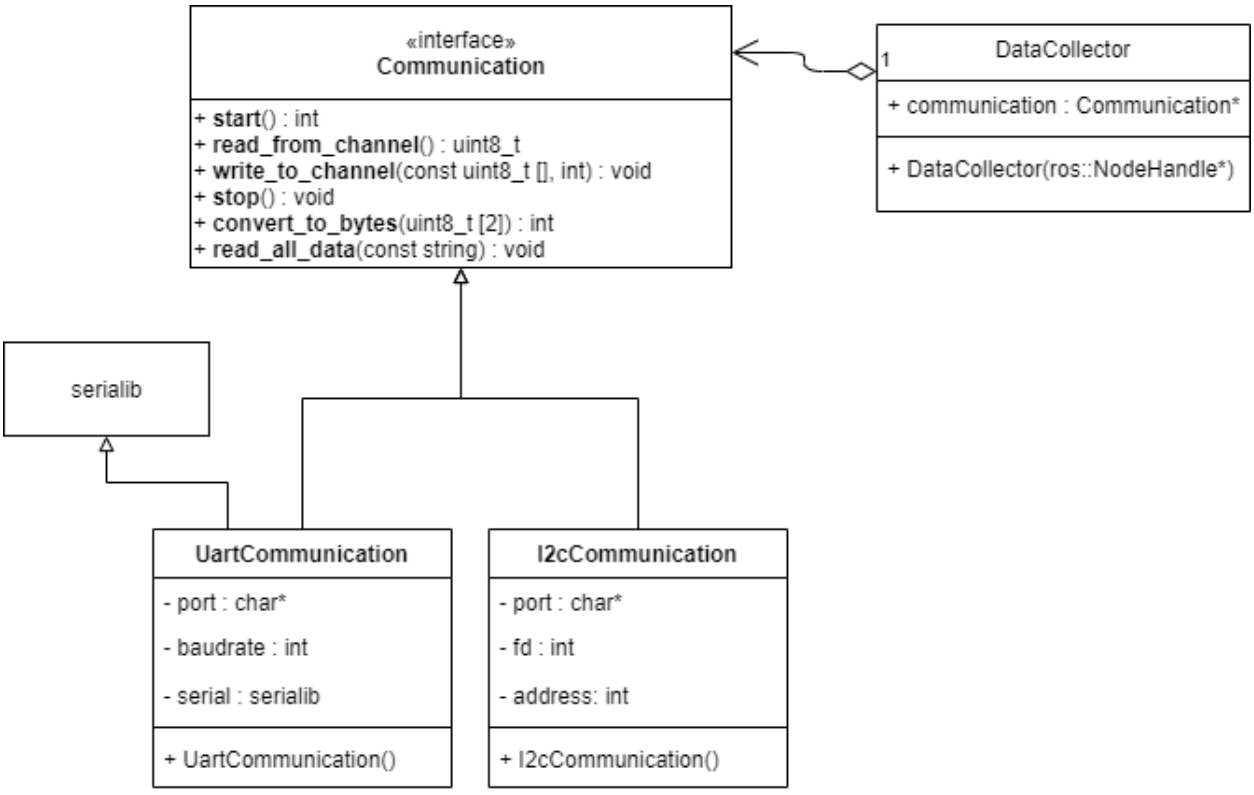

Figure 1: UML diagram of the project

There are 2 possible ways for the sensor to communicate with the sensor: UART or I2C. Communication interface is a generic class the defines the pure virtual functions required to establish the connection. Both communications implement the methods inherited, so inside “ DataCollector ” class, there is only one pointer to a ” Communication ” object.

Functionality & “use-case” scenario

UART communication is based on sending requests and receiving responses. These 2 are represented as byte arrays. For example, to read the “CHIP ID” value (which is A0), the following array should be written to the channel: {0xAA, 0x01,0x00,0x01}. The values from the array represent the start byte (always “AA”, the operation: “read”, register address respectively the number of bytes to be read).

I2C communication is done almost like UART. When we want to read/write data from/to the channel, we only write the slave address (usually 0x28 but depends on the system), the register address and, in case it is a write request, the data to be written.

Before running the project, the user has to setup the parameters in “/config/my_params.yaml” file. Then, the project can be launched using the “roslaunch <package name> <name_of_the_launch_file>” command. For example:

```
$ roslaunch orient_sensor orient_sensor.launch
```

When the project is launched, a console message will appear on the screen and the user will have to choose the type of communication (“0” for UART and “1” for I2C). The baud rate and port name are taken from the param file.

The functionality of each method from “Communication.h” is described in Table 1:

Function name	Input	Output	Description
start()	None	Integer	Opens the communication channel
stop()	None	None	Closes the communication channel
convert_to_bytes(p)	p = Array of 2 bytes	Integer	Utility function used to convert the response from the channel into integer
read_from_channel()	None	Byte (uint8_t)	Reads bytes from the channel. Returns the last byte read
write_to_channel(p1,p2)	p1 = Array of bytes. p2 = Size of the array.	None	Writes bytes to the channel
read_all_data(p)	p = the abbreviation of the data to be displayed. Possible values (const strings): ACC MAG GYR EUL QUA LIA	None	Gets the data according to the given input. Calls “read_from_channel” & “write_to_channel” methods, using the corresponding byte array for each type of data. Prints integers.

	GRV		

The function responsible with each data extraction is “read_all_data(const string opt)”, where “opt” is the name of the data (“ACC”, “MAG” etc). The user declares a “DataCollector” object and instantiates the communication depending on the choice (UART or I2C). Then, the user calls the previous function. For example, to get the acceleration , the following line of code is written: “<DataCollectorObj>.communication -> read_all_data(“ACC”)”. The abbreviation for each type of data is like in the datasheet.