Lab 0x05: I²C and Inertial Measurement Units

This penultimate project should take one week and is to be completed in teams as with previous lab assignments; however, you will be allowed three weeks to complete the assignment so that you can continue working on your term project in parallel with this lab.

If you have been doing a diligent job of writing modular code that is well organized it should be relatively simple to integrate the IMU within your existing user interface from Lab 0x03. On the other hand, this lab may be an "opportunity" to fix any disorganization or flaws in coding structure from past labs if such problems do exist.

In this assignment you will communicate with an IMU over I²C. You can find the datasheet for the BNO055 here.

1 The Project

At this point in the quarter you should be well versed in familiarizing yourself with new hardware and driver libraries. In addition to reading through the datasheet linked above, review the documentation for the `pyb.I2C` driver class. As usual, begin with some experimentation at the REPL before writing any code in a file.

1.1 BNO055 Driver Class

Develop a driver class that will allow you to interface with the BNO055 sensor. Your driver must include the following features, some of which will only make sense after reading through the datasheet:

- An initializer that takes in a `pyb.I2C` object preconfigured in `CONTROLLER` mode[1].
- A method to change the operating mode of the IMU to one of the many "fusion" modes available from the BNO055.
- A method to retrieve the calibration status byte from the IMU and parse it into its individual statuses.
- A method to retrieve the calibration coefficients from the IMU as an array of packed binary data.
- A method to write calibration coefficients back to the IMU from pre-recorded packed binary data.
- A method to read Euler angles from the IMU to use as measurements for feedback.
- A method to read angular velocity from the IMU to use as measurements for feedback.

[1] Historically the term "master" has been used to refer to the primary device on an I2C bus, however we will adopt the word "controller" in this course moving forward.

1.2 IMU Task and Changes to your Existing Tasks

For this lab you will also need create an IMU task to integrate into your existing code structure from Lab 0x03.

You will need to adjust your existing tasks to provide the following modified functionality:

- Add additional single-character commands to your user interface to retrieve data from the IMU. Specifically, add a new command to print the Euler angles retrieved from the IMU and another new command to print the angular velocities retrieved from the IMU. You may choose your own characters for these new commands.

  For each of these new commands, make sure to print the x-, y-, and z-axis components in the correct order and with sign convention well defined. To fit in well with HW 0x03 you should have the x-axis point forward and the y-axis should point out the left window (if Romi had windows). *By default these coordinates may not match the coordinates from the BNO055*. You can fix this in several ways:

  - Physically attach the BNO055 in a way that aligns the axes as you desire.
  - Use math in your own code to transform the values from the BNO055 to be in the desired coordinate system.
  - Use the re-map features of the BNO055 to specify your desired coordinate system.

  The third method above would be most preferrable, but it is up to your team to choose a technique.

- Add functionality to your IMU and user interface tasks to allow calibration on startup:

  1. When your program begins, check to see if a calibration text file is stored on the pybflash flash storage, perhaps named something like `IMU_cal_coeffs.txt`. You can check in code if the file exists by interpreting output from the function `os.listdir()`. Data should be stored in the file as a single row of 22 comma separated hexadecimal numbers, as shown below:

     ```
     0x0,0x0,0x0,0x0,0x0,0x0,0x0,0x0,0x0,0x0,0x0,0x0,0x0,0x0,0x0,0x0,0x0,0x0,0x0,0x0,0x0,0x0
     ```

  2. If the calibration data file exists, your code should automatically read and parse the file and then write the parsed calibration bytes to the IMU.
  3. If the calibration data file does not exist, your code should facilitate calibration through the user interface. This will involve instructing the user on how to manipulate the top Romi robot to follow the BNO055 calibration procedure. For help with understanding the calibration procedure watch this video from Bosch.

     Once calibration is complete, your code should read the 22 bytes of calibration data from the IMU and then write them to the calibration data file on pybflash so that next time you use your kit you will not have to recalibrate.

Due to some nuances in the way the pybflash "drive" is implemented, you may not actually see the generated text file when you look at the drive contents on your laptop. After a power cycle the files should show up, but a soft-reset with Ctrl-D will not make them visible on your PC.

2 Deliverables

You will need to demonstrate that your driver works properly to your lab instructor. A sufficient demonstration will include the following:

- Demonstrate that your calibration interface works correctly by temporarily removing the calibration file on pybflash before running your code. You will not need to demonstrate the full calibration procedure, but you will need to show that the file can be read from, and written to, correctly.
- Display the IMU Euler angles and angular velocities through PuTTY as you change the orientation of the sensor.
- Have Romi pivot in place until it is facing north.

You will submit your demonstration by recording a brief video to then submit on Canvas. You will not need to write a memo for this assignment.