UC Berkeley, ME100, Spring 2021
Lab 7: Sensors

Plan to complete these tasks during the week of April 12.
Checkoff due on Gradescope by Monday April 26, 11:59pm

Introduction and objectives

The purpose of this lab is to become familiar with the inertial measurement unit in your kit and learn how to read and interpret its measurements. You will:

- Install Micropython packages to enable data from the MPU9250 breakout board to be transferred to the ESP32 via I2C
- Explore the measurement capabilities of the MPU9250 and the frequency with which readings can practicably be logged

Step 1: Set up IMU and obtain first readings

- Download the files mpu9250_new.py, imu.py and vector3d.py from bCourses (Files > Labs > Lab7) and copy them onto the /flash directory of your ESP32.
- Connect the MPU9250 breakout board to the ESP32. Connect the GND pin of the breakout board to GND on the ESP32 board, VCC on the breakout board to the USB 5V supply pin on the ESP32 board[1], and connect the SCL and SDA pins of the two boards.
- Download Lab7_MPU9250.py from bCourses, open it in your text editor and study it to note the syntax for reading data from the MPU9250. Then run the script from `shell49`. Enter the REPL and look at the data being printed to the screen every 1 second. You may need to experiment with the "chip ID" until your code works; three likely values are included in a comment in the script.
- Are the temperature values being logged an accurate representation of your room temperature? If not, why not?
- Move the board around and observe how the six inertial readings change. Are all axes of the IMU functioning as you expect? (You may need to increase the sampling rate or change the format in which readings are printed to make clear observations.) **Unfortunately, some of the IMU units may have some axis to be faulty. The faulty axis will appear to be frozen at 0.00 or -1.99… or -1.98… Some will have all axis frozen, or just one or two of the axes. Please check with Tom if you are near campus and pick up a replacement IMU.**
- Make and upload a sketch showing the relationships between the orientations of rotational velocities given by `gyro.{x,y,z}` and the directions of acceleration given by `accel.{x,y,z}`. Show the axes on a sketch of the breakout board. This sketch will be useful to you later on if you use the IMU in your project, as you will know how the recorded data correspond to the orientation of the physical board.
- Observe the values of `mag.{x,y,z}`. If you can move the IMU close to a cable carrying electricity from a wall socket to an appliance (e.g. a desk lamp or computer charger), do you see any changes in the magnetic readings? (Obviously, do not connect any part of your circuit electrically to the wall supply!)

---

[1] Before connecting the 5 V supply, check that the solder jumper near the on-board voltage regulator has *not* been bridged. If it has been bridged, a 3.3 V supply drawn from the "3V" pin on the ESP32 should instead be connected to VCC on the MPU9250 breakout board. See http://www.hiletgo.com/ProductDetail/1953399.html.

Step 2: Log readings locally

The MPU9250 (datasheet on bCourses) claims a sampling rate of up to 8 kHz. Here we will see how frequently we can log readings of a single variable from the device.

- Download Lab7_MPU9250_print.py from bCourses, examine it, and run it from `shell49`. The script makes 100 consecutive readings of *x* acceleration, appending them to an array, and then prints the array's contents to standard output (i.e., the screen) when finished. Recording the data first and then printing them — rather than printing after every reading — helps to reduce the time needed per iteration of the data collection loop.
- What is the time resolution with which readings of this single variable can be obtained? How does this rate compare to the 8 kHz stated sampling limit of the MPU9250?
- Can you think of anything else that could be done to increase the sampling rate on the ESP32?
- What happens if you try to log more than one inertial variable in each iteration of the loop?

If at any stage you get an error message that the I2C bus is already in use, then either reset the ESP32 or enter the REPL and execute `i2c.deinit()`. Remember to exit the REPL before running your next `shell49` command.

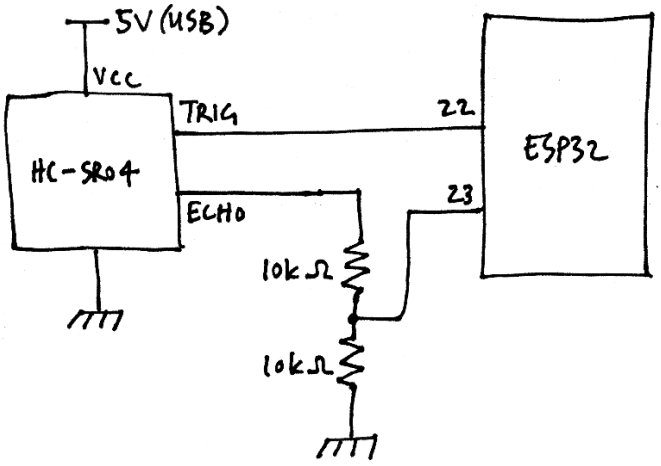
Optional extensions

Step 3: Set up ultrasound sensor

Download the hcsr04.py file and the script to run the sensor, Lab7_ultrasound_ranging.py, from bCourses (Labs > Lab 7). Copy hcsr04.py onto the flash of your ESP32.

Connect the HCSR04 (datasheet here) to your ESP32. There are four connections on the sensor:

- Vcc -- connect to the 5 V "USB" pin on your ESP32.
- Gnd – connect to the GND pin of your ESP32
- Trig – connect this input to an output pin of the ESP32 (GPIO pin 22 is used in the example code linked above). A 3.3 V output from the ESP32 will be read as a logic 1 by the HCSR04.
- Echo – this output needs to be connected *via a voltage divider* to an input pin of the ESP32 (e.g. GPIO pin 23). The HCSR04 generates a 5 V output which may damage the ESP32 if connected directly. Halving the output voltage would be suitable, as a 2.5 V signal will be interpreted by the ESP32 as a logic 1. Two 10 kΩ resistors would make a suitable divider; if you do not have two resistors of the same value you can use a combination of different resistors provided that the highest voltage input to the ESP32 will not exceed 3.3 V.



Run `Lab8_ultrasound_ranging.py` from `shell49`; detected distances will be printed to the terminal display as quickly as the system is able to make readings.

Step 4: Characterize sensor

Point the sensor at a variety of objects (ceiling, wall, table, etc) and confirm that the magnitudes of the distance readings (in centimeters) are roughly as you would expect. Can you find any objects that do not give a realistic reading (e.g. perhaps very rough and/or soft objects that may not reflect strongly or may scatter the reflected ultrasound)?

Then, take a flat, solid object such as a book and move gradually away from the sensor, recording the distance readings from the sensor and measuring the actual corresponding distances with a ruler or tape measure. Make about ten readings over a range of a few cm to about a meter.

Plot the distance returned by the ultrasound sensor against the distance measured with the ruler or tape measure. Do they agree? What is the maximum percentage error you found between the two measurement methods? Are errors systematic or apparently random?

Step 5: Load cell

You kit includes a small load cell rated up to 1 kg force, with an accompanying 24-bit analog to digital converter based on the HX711 chip (datasheet on bCourses). Start by copying hx711.py (available on bCourses) to the flash on your ESP32:

```
cp hx711.py /flash/hx711.py
```

Next, you will need to connect the wires on your load cell to the HX711 board. The connections need to be:

- Red wire (Positive excitation) to E+ on the HX711 board
- Black wire (negative excitation) to E–
- Green wire to A+
- White wire to A–

Ordinarily these wires would be soldered directly onto the board, but since you may well not have access to a soldering iron, you may need to improvise connections between the headers and the load

cell wires, possibly using the jumper wires in your kit and by twisting wires to connect them. Unfortunately, the HX711 board is too wide to be placed on your protoboard while leaving space for wires also to be inserted.

Then connect:

- VCC on the HX711 board to the “USB” 5V supply pin on the ESP32 board,
- GND on the HX711 board to GND on the ESP32 board,
- The data pin, probably labeled “DT”, to GPIO pin 4
- The clock pin, SCK, to GPIO pin 5.

Download Lab7_HX711.py from bCourses and run this using `shell49`. Enter the REPL and you will see a numerical value being printed every 0.1 seconds. Gently flex the load cell and note how the reading changes. If you plan to use this in your project, the relationship between load and recorded value will need to be calibrated experimentally.