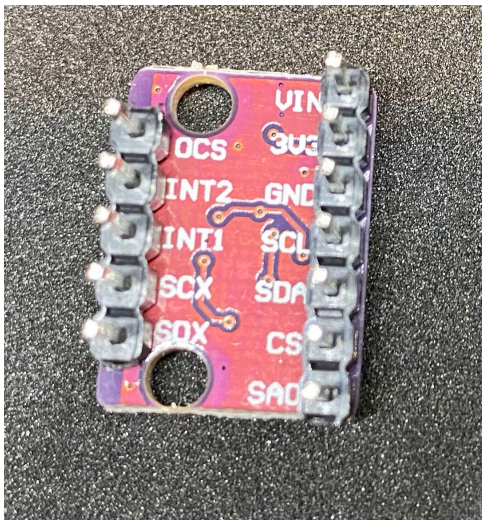
21M.370 Digital Instrument Design

IMU and WiFi tutorial

This document covers how to use the accelerometer that comes with your kit, and how to use WiFi to stream data to python.

LSM6DS3

The LSM6DS3 is a dual 3-axis accelerometer and gyroscope, and also has a temperature sensor (normally used to calibrate the IMU).



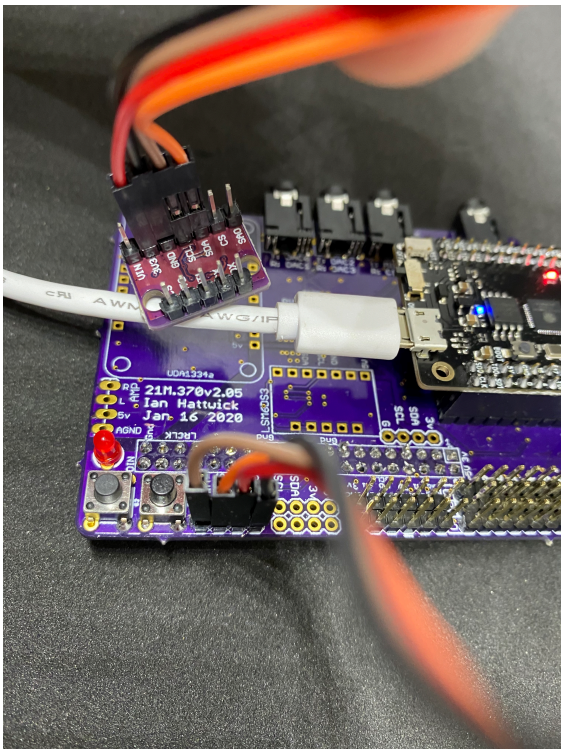
The LSM6DS3 can connect to your ESP32 using either I2C (like we are doing for the MPR121) and also using a different protocol, SPI. For now, we will stick with using I2C.

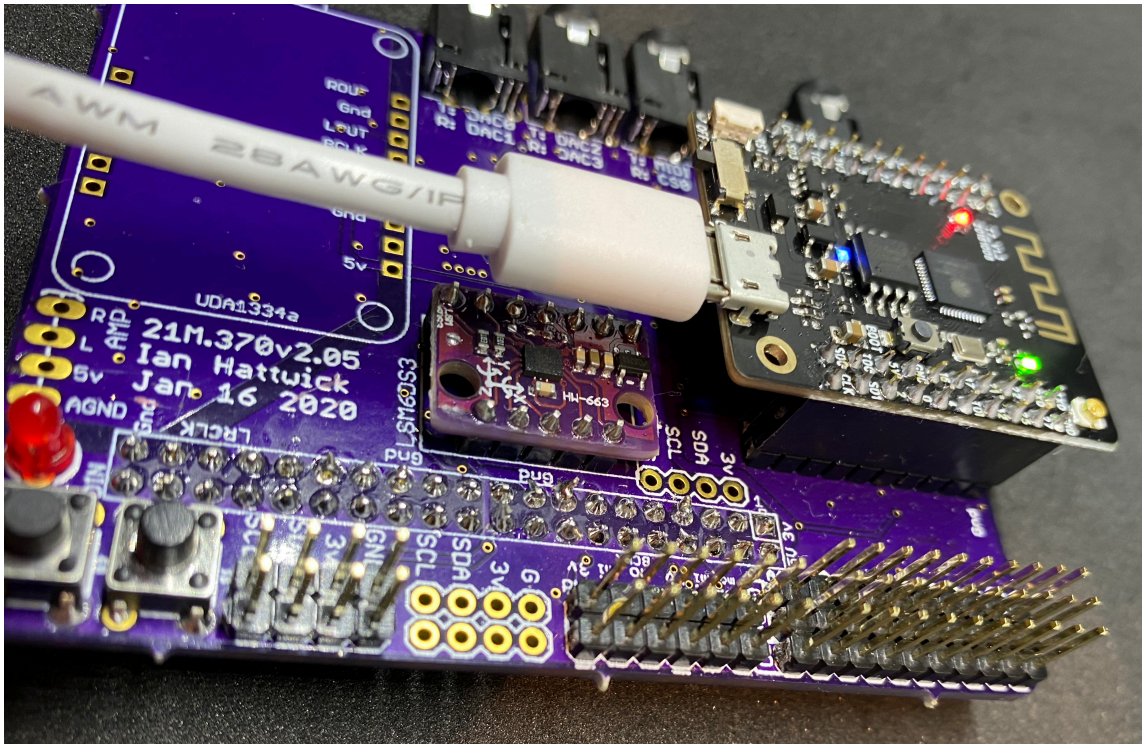
To start, solder pins onto the LSM6DS3 breakout, making sure they are sticking out the correct way. Go ahead and solder all of the pins, even though we will only use 4.

Connect the pins to the ESP32 just like we did for the MPR121, making sure to match the SDA and SCL pins. I suggest using a cable to connect it the first time to test and make sure it works.

I2C is designed to allow multiple devices on the same bus, so you should be able to connect the MPR121 and LSM6DS3 at the same time.

There is also a footprint on the PCB to solder the IMU directly to the PCB. I'd test it using a cable first :-) But soldering the IMU to the PCB will give it a strong mechanical coupling, which will help improve the quality of the accelerometer data if PCB movement is what you are trying to detect.





Using WiFi to monitor the accelerometer

Once the IMU is plugged in, open M370_ACCEL.ino in Arduino. This is further development of our primary M370 file, but adds support for WiFi as well as serial communication. You will have to choose which method you want to use by setting SERIAL_ENABLE or WIFI_ENABLE to 1, but not both at the same time. If you enable wifi you will also have to add your SSID (the wifi network name) and your wifi password.

```
#include <WiFiUdp.h>

byte SERIAL_ENABLE = 0; //enables communication over USB
byte WIFI_ENABLE = 1; //enables communication over USB

// WiFi network name and password:
const char * ssid = "MLE";
const char * password = "mitmusictech";
```

The matching python script is ’370_accel.py’. You will also have to choose wifi or serial in this script on line 30/31.

```
26  ######################
27  # SET COMMUNICATION MODE
28  ######################
29  # don't forget to set the ESP32 firmware to match!
30  SERIAL_ENABLE = 0
31  WIFI_ENABLE = 1 #!!!! READ FOLLOWING COMMENT
32  # !!!! WIFI MAY require that you reset ESP32 before running python script
33  # !!!! and DOES require that you run the python script AFTER resetting the ESP32
```

After that, you can enable the accelerometer or gyro just like an analog input, with the same options:

‘address’: the osc address received in PD

‘enable’: toggle whether this data is sent

‘data rate’: how fast (in ms) the data is sent to python by the ESP32

‘mode’: how the ESP32 processes sampled data before sending to python.

- the ESP32 samples the data multiple times between sending it
- The available mores are: MEAN, MEDIAN, MIN, MAX, PEAK_DEVIATION

- The number of samples is set in the ESP32 script. Below is the default setting, which samples 8x for each value sent to python:

```
/*********************************************
IMU SETUP
*********************************************/
int  imuInterval = 100;

IMUclass accels[3] =  {
  IMUclass( 0,"accelX", imuInterval, 8, MEAN ),
  IMUclass( 1,"accelY", imuInterval, 8, MEAN ),
  IMUclass( 2,"accelZ", imuInterval, 8, MEAN )
};
```

Console messages

With wifi enabled, the console will print out the status of the connection process. It should look something like this:

```
checking WiFi. . .
error [Errno 35] Resource temporarily unavailable
checking WiFi. . .
received message: b'interval: 100' address ('192.168.1.8', 1234) length 13
Wifi connected to  ('192.168.1.8', 1234)
Sending data to port 5005
```

The message which indicates successful connection to the ESP32 is:
‘Wifi connected to (IP address of the ESP32, port)’
‘Port 5005’ is the port used for sending data from python to PD.

In pure data

Two new patches in pure data will display the raw data and also demonstrate some common signal processing techniques.

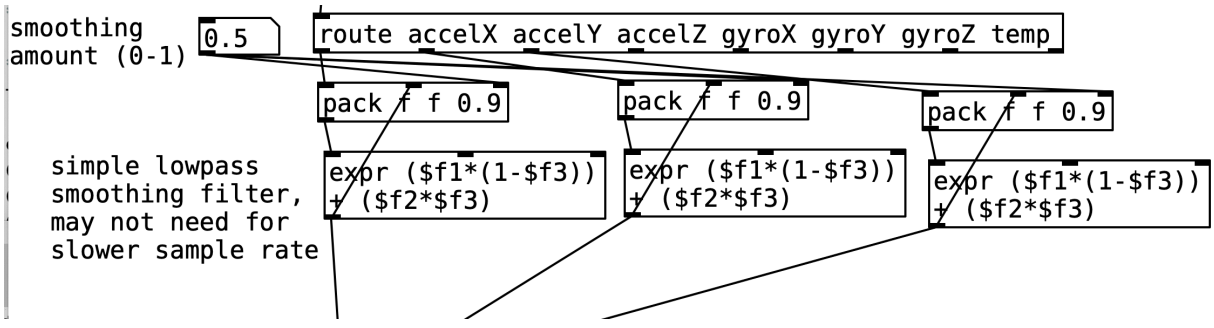
accelMonitoring

This patch simply monitors the incoming data from the accelerometer, gyroscope, and temperature sensor. Be sure to use the toggle to enable monitoring.

accelSignalProcessing

This patch uses a couple of standard processing techniques to get useful information from the accelerometer data.

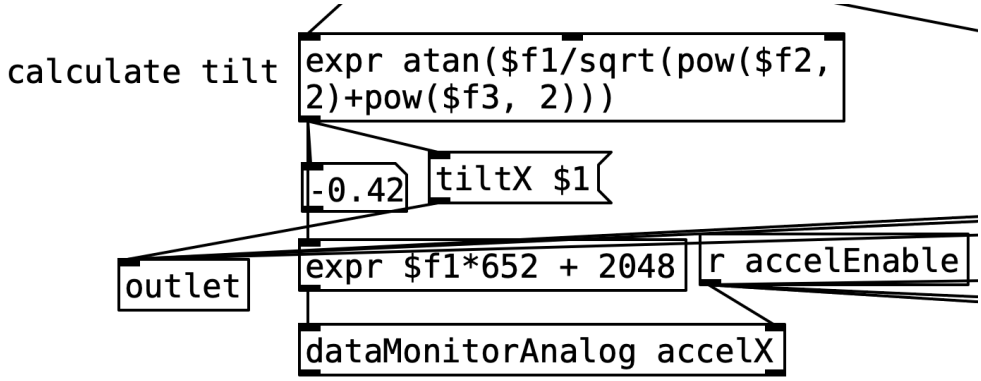
Lowpass filter
This simple one-pole lowpass filter uses a single coefficient. This filter adds together the most recent signal data with the previous output of the filter in a feedback configuration. The smoothing amount weights how much of the output of the filter is the new data and how much is the previous data. With values close to 1, new data has less influence on the output value which slows the response of filter to fast changing data (hence the lowpass name).



Tilt processing

The accelerometer outputs sensitivity to acceleration for all three axes, which is often used as a tilt control due to the changing effect of gravity as the accelerometer is

rotated. While just using each axis on its own as a tilt control is common, a more accurate method combines the data from all three axes to generate tilt for X, Y, and Z.



The formula for one tilt axis is in the first expr object:
atan($f1 /sqrt(pow($f2,2) + pow($f3,2)))

The second expr object just scales the data for display.

Magnitude

Another common piece of information we might want is total acceleration of all three axes, or the magnitude of the acceleration vector. To calculate that we take the cube root of the sum of all three axes.