21M.370 Digital Instrument Design
Lab assignment 7

This lab assignment describes an instrument which uses an IMU as one of its primary inputs. The basic instrument is called Chester, and build instructions, ESP32 firmware, python scripts, and PD patches are provided.

This is an optional lab assignment - provided as a potential basis for your final project. But whether you build Chester or not you should add the IMU to your board (as described in the videos below)! In addition, the WiFi notes below can be useful if you'd like your controller to be wireless.

In addition to the basic Chester scripts provided you can also add modifications. For example, you can use the IMU data to control additional synthesizer parameters. You can use any of the existing IMU signals (raw/smoothed/magnitude of accelerometer, raw gyroscope, tilt, velocity, jerk, velocity magnitude) or create your own IMU signal. Some good targets for mapping IMU data:

- use jerk to trigger a burst of reverb
- control the waveshape of BWL-OSC 2, 3, and 4. (The waveform of BWL-OSC 1 is different and responds to waveshape in a very different manner).
- Change the REVERB time of the two reverbs
- Change the pitch glide time
- Add additional modules (oscillators, effects, etc.) and control those.

Second, you can map the encoder to control parameters. In the mapping tutorial I show how to use it to control the master volume, and also the smoothing for the magnitude value - and you can follow that example, or use the encoder in a different way. Using the encoder to change smoothing

or scaling for input signals is a great way to be able to tweak the feel of your instrument on the fly.

Building Chester

1. Assemble 4 lever switches, 1 encoder, and the IMU module.
2. Test the IMU module with your ESP32 breakout board, and when you have confirmed it is working solder it directly to the breakout PCB.
3. Connect the switches to p6-p9, the encoder switch to p10, and the encoder A and B to p12 and p13. DON'T USE p11 for the encoder! Due to a quirk of the ESP32 it won't work in this instance.
4. Assemble Chester, either following my example or creating your own vision.
5. Play with the provided python script and PD patches to learn the instrument.
6. Decide how you would like to add additional IMU controls as well as how you will use the encoder. Implement these in your python script following the mapping video example.

Video tutorials

1. Soldering the sensors
2. Assembling Chester
3. Testing the sensors in firmware
4. Overview of working with IMUs
5. Mapping tutorial
6. Chester demo video
7. Chester overview

Links:

Accelerometer and Gyro guide by Sparkfun

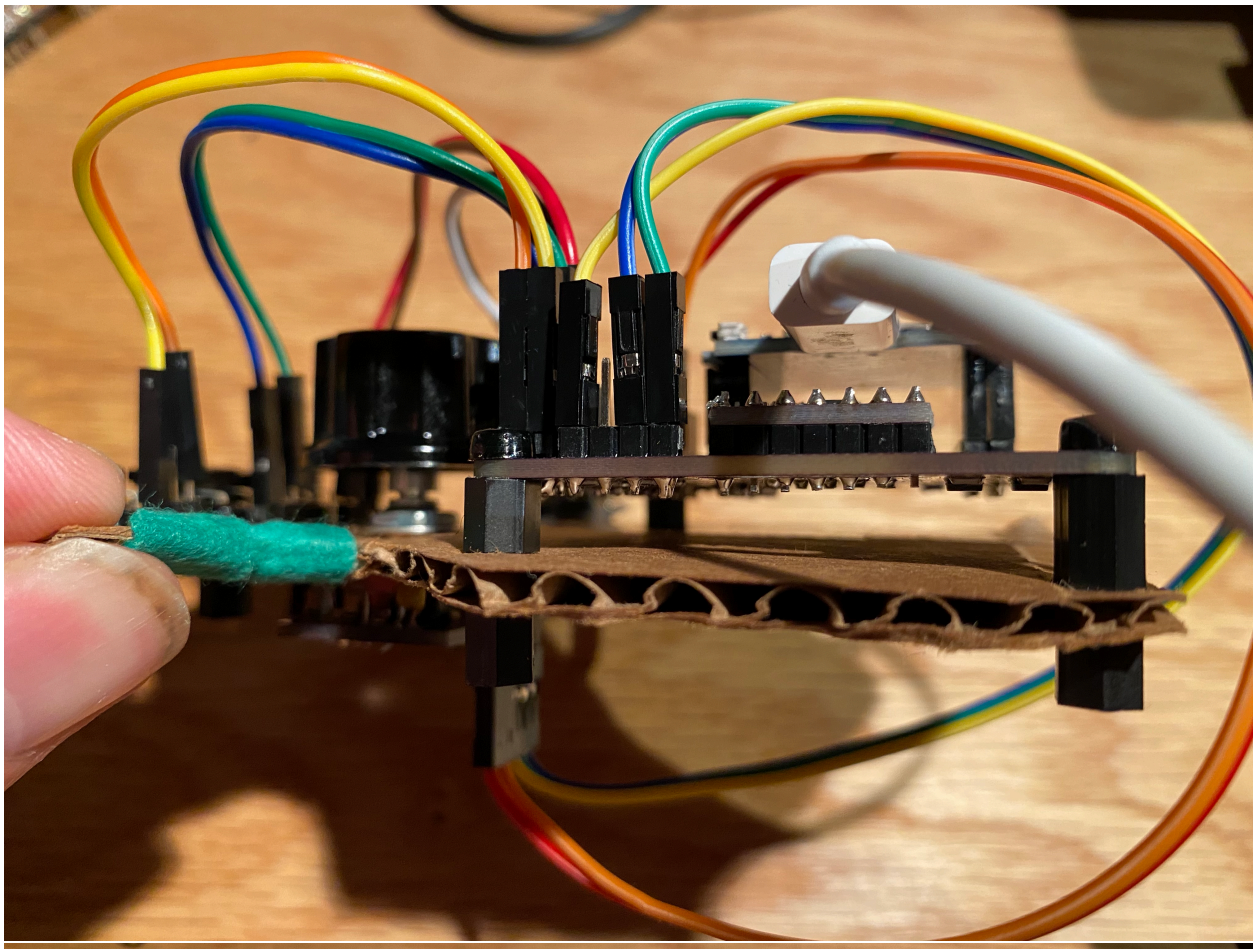
Practical Guide to Kalman filters for sensor fusion

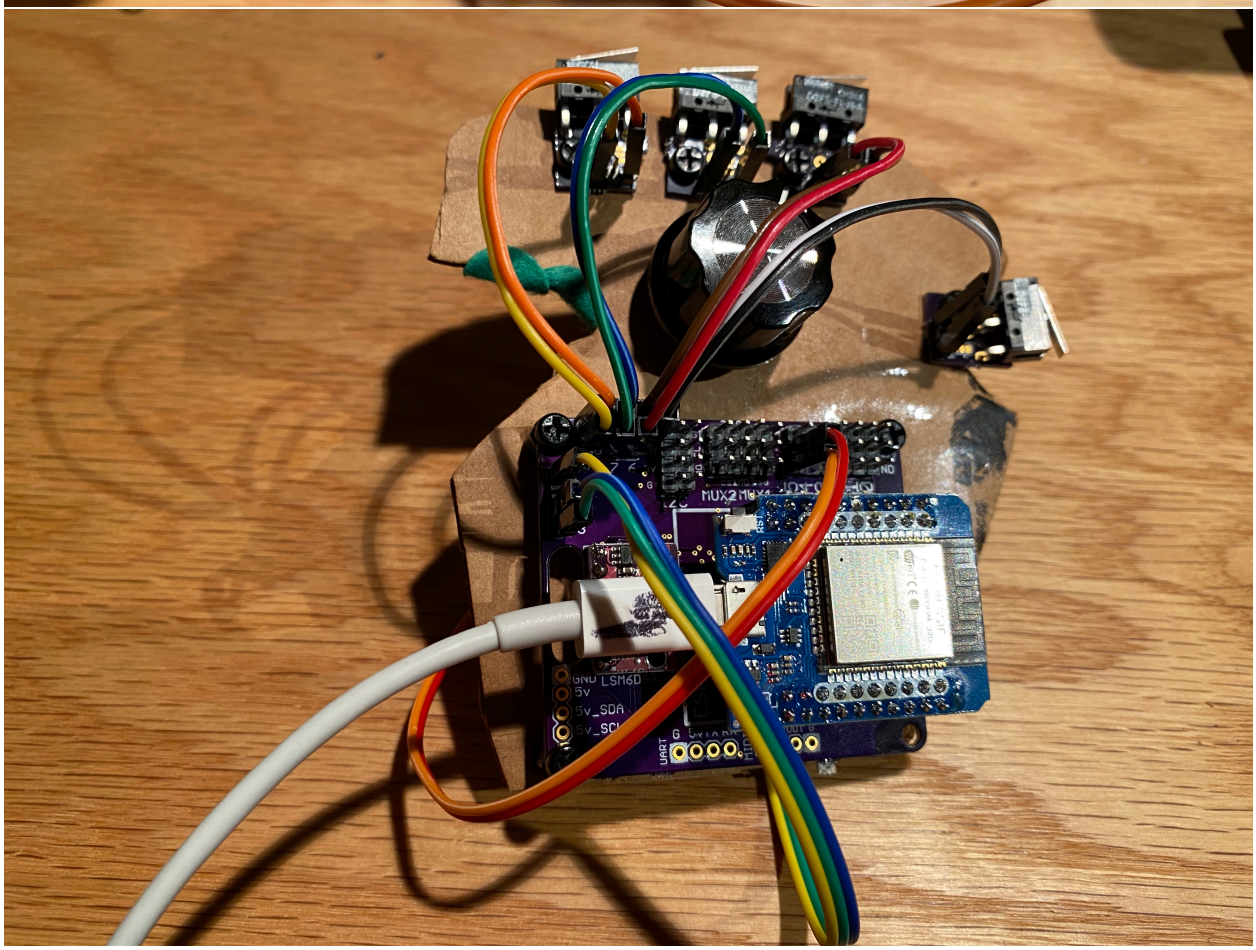
Reading a IMU Without Kalman: The Complementary Filter

Notes on Chester build:

1. Make sure the lever switches are attached to pins 6-9
2. Make sure the encoder switch is on pin 10, and the encoder A and B pins are on pins 12 and 13. You should not use pin 11 for this project.
3. You can follow the same design as I created for chester (a handheld instrument built with a cardboard frame) or make your own in any shape you would like. Obviously, you need to make something that will allow you to use lots of motion!
4. When attaching the PCB to the cardboard frame, be sure to use standoffs as in the pictures below, which is slightly different from how I did it in the video. You should use at least three standoffs so the board is rigidly attached to the frame.
5. You can disable IMU monitoring at the top of oscMapping.py. This is a good idea if you find your computer struggling. We are sending lots of data from Python to PD!
6. If you really need to slow down the data being sent to python, you can also change the IMU data send rate. In the file 'IMU.ino' (which should appear as a tab in Arduino) line 39 sets the interval in milliseconds at which we send IMU data to python. 50ms is actually not that fast - but you could try slowing it down and seeing if it helps. Note we are actually reading the data every 5ms, and then sending the average value to Python.

```
38  static uint32_t sendTimer = 0;
39  int interval = 50;
40
41  if(millis()-sendTimer>interval){
42    sendTimer=millis();
43
```




MUX2
GND LSM6D
5v
5v_SDA
5v_SCL
UART