import time

import random

import math

import threading

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| |  | | --- | | class HX711: | |  | def \_\_init\_\_(self, dout, pd\_sck, gain=128): | |  | self.PD\_SCK = pd\_sck | |  |  | |  | self.DOUT = dout | |  |  | |  | # Last time we've been read. | |  | self.lastReadTime = time.time() | |  | self.sampleRateHz = 80.0 | |  | self.resetTimeStamp = time.time() | |  | self.sampleCount = 0 | |  | self.simulateTare = False | |  |  | |  | # Mutex for reading from the HX711, in case multiple threads in client | |  | # software try to access get values from the class at the same time. | |  | self.readLock = threading.Lock() | |  |  | |  | self.GAIN = 0 | |  | self.REFERENCE\_UNIT = 1 # The value returned by the hx711 that corresponds to your reference unit AFTER dividing by the SCALE. | |  |  | |  | self.OFFSET = 1 | |  | self.lastVal = long(0) | |  |  | |  | self.DEBUG\_PRINTING = False | |  |  | |  | self.byte\_format = 'MSB' | |  | self.bit\_format = 'MSB' | |  |  | |  | self.set\_gain(gain) | |  |  | |  |  | |  |  | |  |  | |  | # Think about whether this is necessary. | |  | time.sleep(1) | |  |  | |  | def convertToTwosComplement24bit(self, inputValue): | |  | # HX711 has saturating logic. | |  | if inputValue >= 0x7fffff: | |  | return 0x7fffff | |  |  | |  | # If it's a positive value, just return it, masked with our max value. | |  | if inputValue >= 0: | |  | return inputValue & 0x7fffff | |  |  | |  | if inputValue < 0: | |  | # HX711 has saturating logic. | |  | if inputValue < -0x800000: | |  | inputValue = -0x800000 | |  |  | |  | diff = inputValue + 0x800000 | |  |  | |  | return 0x800000 + diff | |  |  | |  |  | |  | def convertFromTwosComplement24bit(self, inputValue): | |  | return -(inputValue & 0x800000) + (inputValue & 0x7fffff) | |  |  | |  |  | |  | def is\_ready(self): | |  | # Calculate how long we should be waiting between samples, given the | |  | # sample rate. | |  | sampleDelaySeconds = 1.0 / self.sampleRateHz | |  |  | |  | return time.time() >= self.lastReadTime + sampleDelaySeconds | |  |  | |  |  | |  | def set\_gain(self, gain): | |  | if gain is 128: | |  | self.GAIN = 1 | |  | elif gain is 64: | |  | self.GAIN = 3 | |  | elif gain is 32: | |  | self.GAIN = 2 | |  |  | |  | # Read out a set of raw bytes and throw it away. | |  | self.readRawBytes() | |  |  | |  |  | |  | def get\_gain(self): | |  | if self.GAIN == 1: | |  | return 128 | |  | if self.GAIN == 3: | |  | return 64 | |  | if self.GAIN == 2: | |  | return 32 | |  |  | |  | # Shouldn't get here. | |  | return 0 | |  |  | |  |  | |  | def readRawBytes(self): | |  | # Wait for and get the Read Lock, incase another thread is already | |  | # driving the virtual HX711 serial interface. | |  | self.readLock.acquire() | |  |  | |  | # Wait until HX711 is ready for us to read a sample. | |  | while not self.is\_ready(): | |  | pass | |  |  | |  | self.lastReadTime = time.time() | |  |  | |  | # Generate a 24bit 2s complement sample for the virtual HX711. | |  | rawSample = self.convertToTwosComplement24bit(self.generateFakeSample()) | |  |  | |  | # Read three bytes of data from the HX711. | |  | firstByte = (rawSample >> 16) & 0xFF | |  | secondByte = (rawSample >> 8) & 0xFF | |  | thirdByte = rawSample & 0xFF | |  |  | |  | # Release the Read Lock, now that we've finished driving the virtual HX711 | |  | # serial interface. | |  | self.readLock.release() | |  |  | |  | # Depending on how we're configured, return an orderd list of raw byte | |  | # values. | |  | if self.byte\_format == 'LSB': | |  | return [thirdByte, secondByte, firstByte] | |  | else: | |  | return [firstByte, secondByte, thirdByte] | |  |  | |  |  | |  | def read\_long(self): | |  | # Get a sample from the HX711 in the form of raw bytes. | |  | dataBytes = self.readRawBytes() | |  |  | |  |  | |  | if self.DEBUG\_PRINTING: | |  | print(dataBytes,) | |  |  | |  | # Join the raw bytes into a single 24bit 2s complement value. | |  | twosComplementValue = ((dataBytes[0] << 16) | | |  | (dataBytes[1] << 8) | | |  | dataBytes[2]) | |  |  | |  | if self.DEBUG\_PRINTING: | |  | print("Twos: 0x%06x" % twosComplementValue) | |  |  | |  | # Convert from 24bit twos-complement to a signed value. | |  | signedIntValue = self.convertFromTwosComplement24bit(twosComplementValue) | |  |  | |  | # Record the latest sample value we've read. | |  | self.lastVal = signedIntValue | |  |  | |  | # Return the sample value we've read from the HX711. | |  | return int(signedIntValue) | |  |  | |  |  | |  | def read\_average(self, times=3): | |  | # Make sure we've been asked to take a rational amount of samples. | |  | if times <= 0: | |  | print("HX711().read\_average(): times must >= 1!! Assuming value of 1.") | |  | times = 1 | |  |  | |  | # If we're only average across one value, just read it and return it. | |  | if times == 1: | |  | return self.read\_long() | |  |  | |  | # If we're averaging across a low amount of values, just take an | |  | # arithmetic mean. | |  | if times < 5: | |  | values = int(0) | |  | for i in range(times): | |  | values += self.read\_long() | |  |  | |  | return values / times | |  |  | |  | # If we're taking a lot of samples, we'll collect them in a list, remove | |  | # the outliers, then take the mean of the remaining set. | |  | valueList = [] | |  |  | |  | for x in range(times): | |  | valueList += [self.read\_long()] | |  |  | |  | valueList.sort() | |  |  | |  | # We'll be trimming 20% of outlier samples from top and bottom of collected set. | |  | trimAmount = int(len(valueList) \* 0.2) | |  |  | |  | # Trim the edge case values. | |  | valueList = valueList[trimAmount:-trimAmount] | |  |  | |  | # Return the mean of remaining samples. | |  | return sum(valueList) / len(valueList) | |  |  | |  |  | |  | def get\_value(self, times=3): | |  | return self.read\_average(times) - self.OFFSET | |  |  | |  |  | |  | def get\_weight(self, times=3): | |  | value = self.get\_value(times) | |  | value = value / self.REFERENCE\_UNIT | |  | return value | |  |  | |  |  | |  | def tare(self, times=15): | |  | # If we aren't simulating Taring because it takes too long, just skip it. | |  | if not self.simulateTare: | |  | return 0 | |  |  | |  | # Backup REFERENCE\_UNIT value | |  | reference\_unit = self.REFERENCE\_UNIT | |  | self.set\_reference\_unit(1) | |  |  | |  | value = self.read\_average(times) | |  |  | |  | if self.DEBUG\_PRINTING: | |  | print("Tare value:", value) | |  |  | |  | self.set\_offset(value) | |  |  | |  | # Restore the reference unit, now that we've got our offset. | |  | self.set\_reference\_unit(reference\_unit) | |  |  | |  | return value; | |  |  | |  |  | |  | def set\_reading\_format(self, byte\_format="LSB", bit\_format="MSB"): | |  |  | |  | if byte\_format == "LSB": | |  | self.byte\_format = byte\_format | |  | elif byte\_format == "MSB": | |  | self.byte\_format = byte\_format | |  | else: | |  | print("Unrecognised byte\_format: \"%s\"" % byte\_format) | |  |  | |  | if bit\_format == "LSB": | |  | self.bit\_format = bit\_format | |  | elif bit\_format == "MSB": | |  | self.bit\_format = bit\_format | |  | else: | |  | print("Unrecognised bit\_format: \"%s\"" % bit\_format) | |  |  | |  |  | |  |  | |  | def set\_offset(self, offset): | |  | self.OFFSET = offset | |  |  | |  |  | |  | def get\_offset(self): | |  | return self.OFFSET | |  |  | |  |  | |  | def set\_reference\_unit(self, reference\_unit): | |  | # Make sure we aren't asked to use an invalid reference unit. | |  | if reference\_unit == 0: | |  | print("HX711().set\_reference\_unit(): Can't use 0 as a reference unit!!") | |  | return | |  |  | |  | self.REFERENCE\_UNIT = reference\_unit | |  |  | |  |  | |  | def power\_down(self): | |  | # Wait for and get the Read Lock, incase another thread is already | |  | # driving the HX711 serial interface. | |  | self.readLock.acquire() | |  |  | |  | # Wait 100us for the virtual HX711 to power down. | |  | time.sleep(0.0001) | |  |  | |  | # Release the Read Lock, now that we've finished driving the HX711 | |  | # serial interface. | |  | self.readLock.release() | |  |  | |  |  | |  | def power\_up(self): | |  | # Wait for and get the Read Lock, incase another thread is already | |  | # driving the HX711 serial interface. | |  | self.readLock.acquire() | |  |  | |  | # Wait 100 us for the virtual HX711 to power back up. | |  | time.sleep(0.0001) | |  |  | |  | # Release the Read Lock, now that we've finished driving the HX711 | |  | # serial interface. | |  | self.readLock.release() | |  |  | |  | # HX711 will now be defaulted to Channel A with gain of 128. If this | |  | # isn't what client software has requested from us, take a sample and | |  | # throw it away, so that next sample from the HX711 will be from the | |  | # correct channel/gain. | |  | if self.get\_gain() != 128: | |  | self.readRawBytes() | |  |  | |  |  | |  | def reset(self): | |  | # self.power\_down() | |  | # self.power\_up() | |  |  | |  | # Mark time when we were reset. We'll use this for sample generation. | |  | self.resetTimeStamp = time.time() | |  |  | |  |  | |  | def generateFakeSample(self): | |  | sampleTimeStamp = time.time() - self.resetTimeStamp | |  |  | |  | noiseScale = 1.0 | |  | noiseValue = random.randrange(-(noiseScale \* 1000),(noiseScale \* 1000)) / 1000.0 | |  | sample = math.sin(math.radians(sampleTimeStamp \* 20)) \* 72.0 | |  |  | |  | self.sampleCount += 1 | |  |  | |  | if sample < 0.0: | |  | sample = -sample | |  |  | |  | sample += noiseValue | |  |  | |  | BIG\_ERROR\_SAMPLE\_FREQUENCY = 142 | |  | ###BIG\_ERROR\_SAMPLE\_FREQUENCY = 15 | |  | BIG\_ERROR\_SAMPLES = [0.0, 40.0, 70.0, 150.0, 280.0, 580.0] | |  |  | |  | if random.randrange(0, BIG\_ERROR\_SAMPLE\_FREQUENCY) == 0: | |  | sample = random.sample(BIG\_ERROR\_SAMPLES, 1)[0] | |  | print("Sample %d: Injecting %f as a random bad sample." % (self.sampleCount, sample)) | |  |  | |  | sample \*= 1000 | |  |  | |  | sample \*= self.REFERENCE\_UNIT | |  |  | |  | return int(sample) | |  |  | |  |  | |  | # EOF - emulated\_hx711.py | |  |