

G54MRT

Python and Sensors 2: Characterising and filtering sensor data

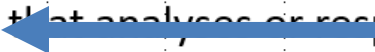
Joe Marshall, Stuart Reeves

Sensor processing for the coursework

- What you need to do technically for the coursework is covered in these four lectures:
- Lecture 1: Get data from sensors
- **Lecture 2: Filter the raw data so that it is usable**
- Lecture 3: Combine two or more data streams to make some kind of inference, such as:
 - Has an event happened (e.g. someone has opened a door, someone has knocked on the door)
 - The value of an unknown quantity (e.g. how many people do we think are in a room, how 'busy' is the room, what shape is the room)
 - What is happening to a device (e.g. is it being thrown around, has it been dropped, is it outside or inside)
- Lecture 4: Test it

Do something clever with it

Your project must use at least **two sensors**.

And do some meaningful processing  that analyses or responds to the data from both of them.

You can't just display sensor data.

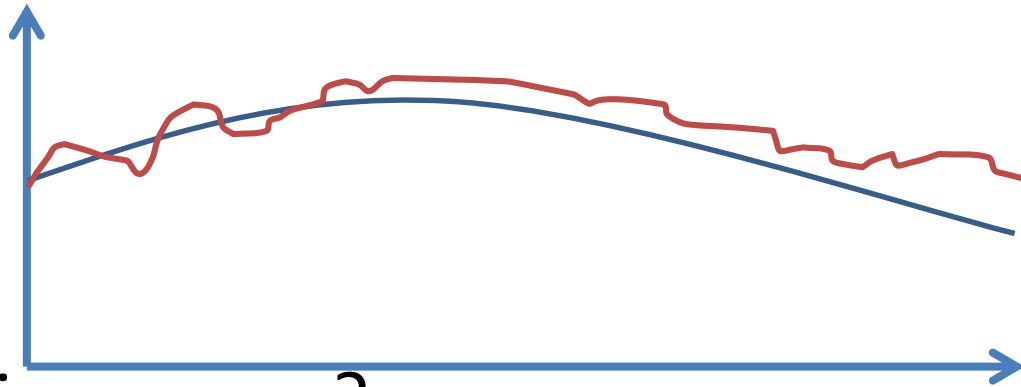
You can (for example):

- Use sensor data to make inferences about things, e.g.
 - What is the state of the room / environment?
 - What is happening to the system (or an object to which it is attached)?
 - Who is using a system?
- Use sensor data as an input device, e.g.
 - Where is something being touched?
 - How hard is something hit?

In the report, you need to demonstrate:

- What you based your sensor algorithms on (real world data, algorithms from research literature ...)
- What your sensor algorithms do?
- Do they work? (demonstrate by running tests and reporting numerical results & statistics)

Characterising sensors



- What is a sensor?
- 'A device that receives a stimulus and responds with an electrical signal'
- Understanding **how** it responds is key to using a sensor
- Read the book chapter on all this stuff on Moodle

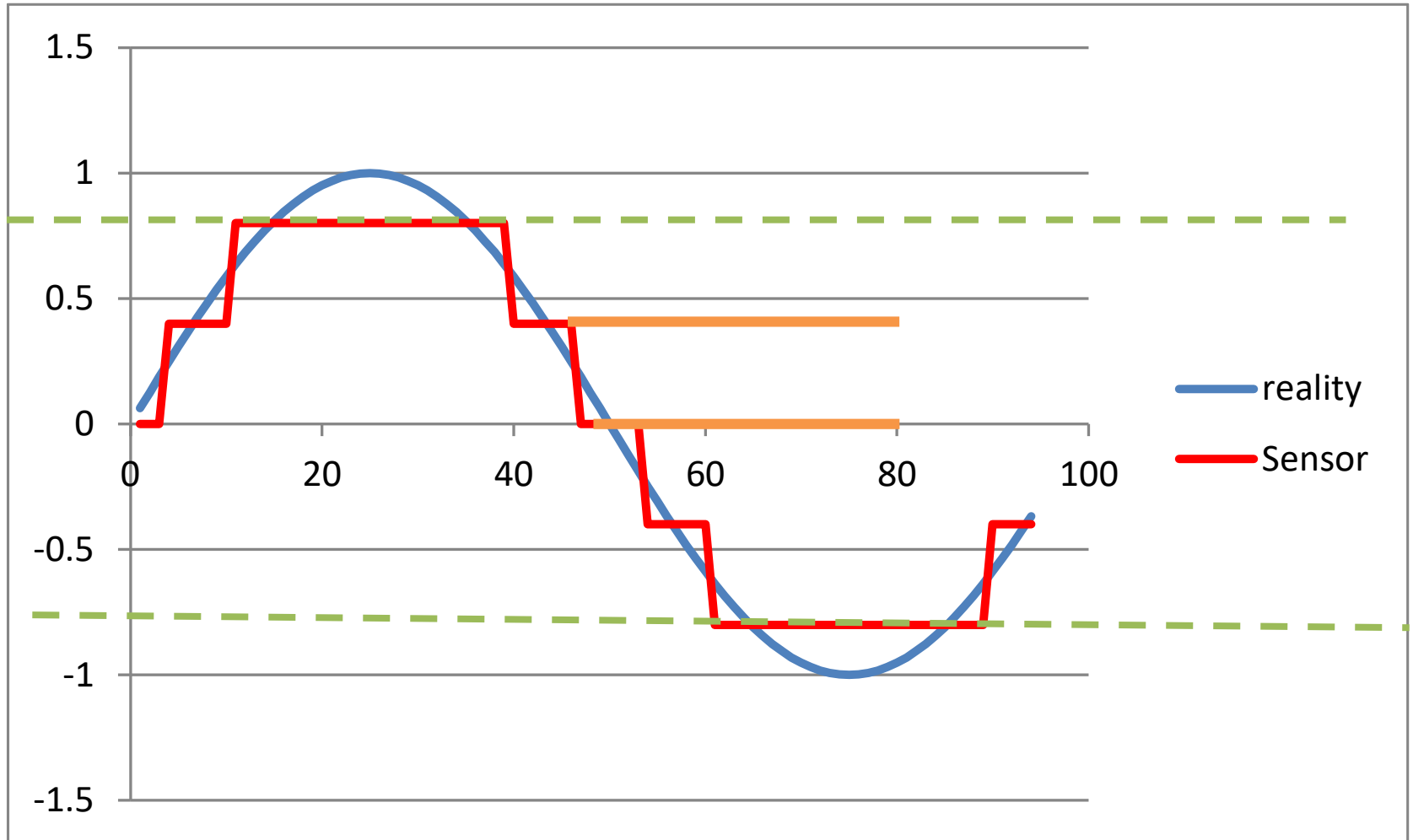
Error

- Sensors have errors
- Difference between the real world value and the value as output by the sensor
- We don't always know the correct value
 - e.g. temperature
- Many types & causes of error (see book chapter on moodle)
- This is about the main ones which are likely to be relevant to the coursework
- And ways to mitigate the effects of error on your algorithms

Range and resolution and sensitivity

- Converting analog to digital
- On GrovePI (or Arduino etc.) 10 bit ADC = 1024 levels.
- Resolution = how many levels over the range
- Range = Min and max of sensor
- Sensitivity of sensor limited to minimum step size
 - Sensitivity = how much real value change makes 1 level change
- The 10 bit grove sensors are sensitive enough for most things you are likely to do in your coursework

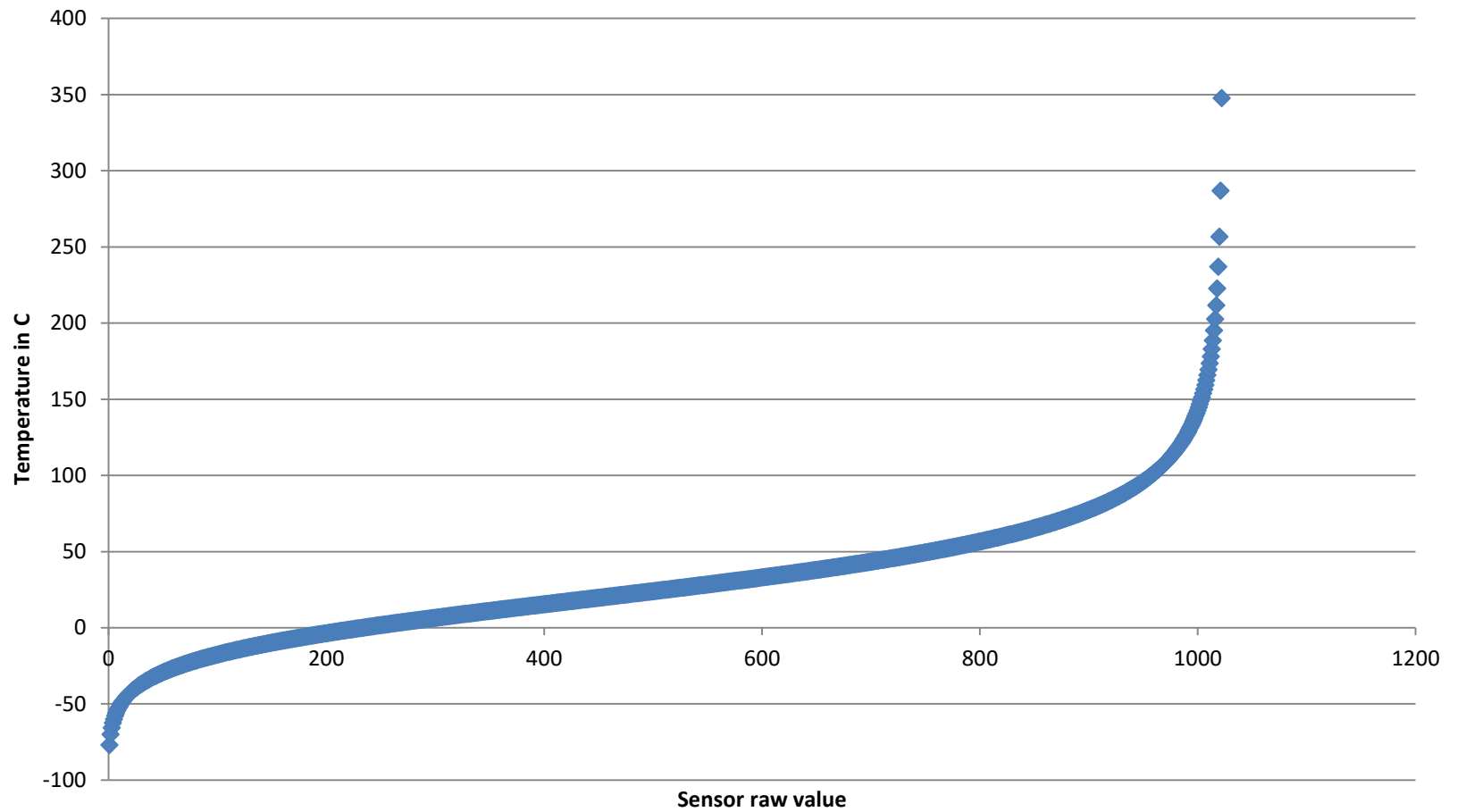
Range and sensitivity



Non-linearity

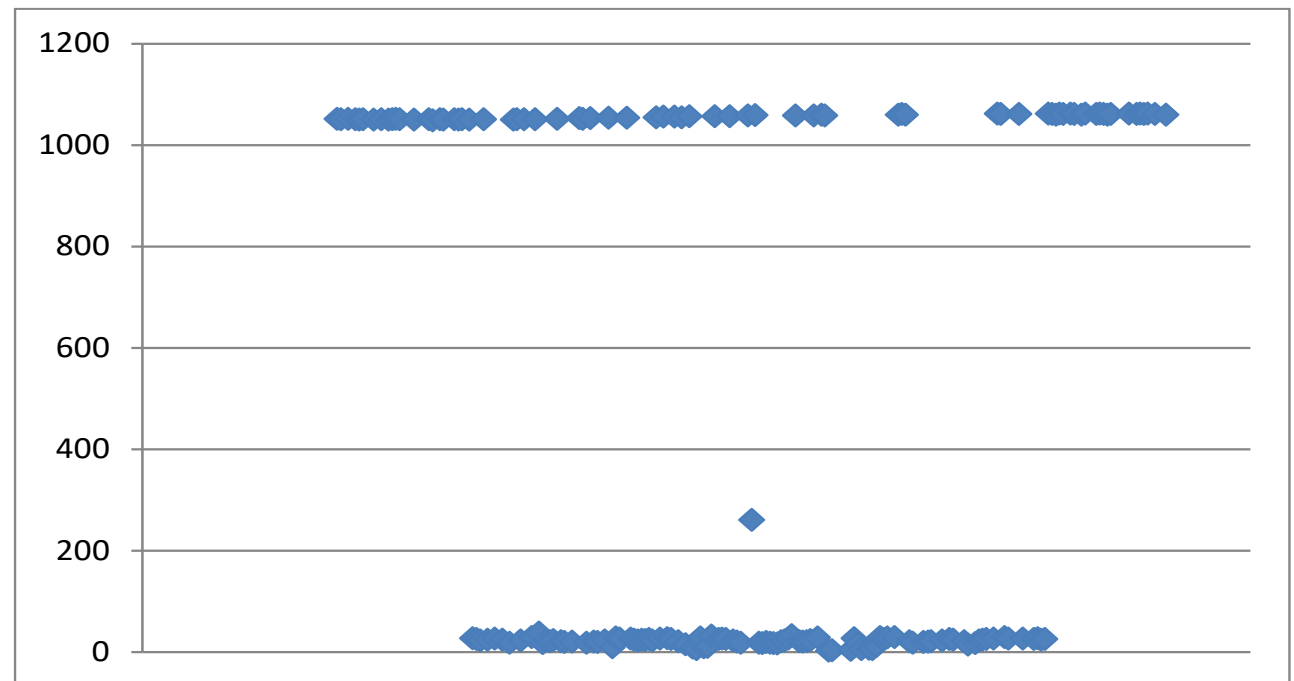
- Raw sensor data = voltage from sensor,
0 = 0V, 1023= 5V
- For some sensors the quantity being measured is related to this via a mapping
- Which may not be linear – sensitivity changes through the range
- For example, temperature sensor = thermistor
 - Resistance changes based on temperature
 - Resistance $R = 10,000 * (1023 - \text{sensorVal}) / \text{sensorVal}$
 - Temperature = $1 / (\ln(R / 10000) / 3975 + 1 / 298.15) - 273.15$

Non-linearity



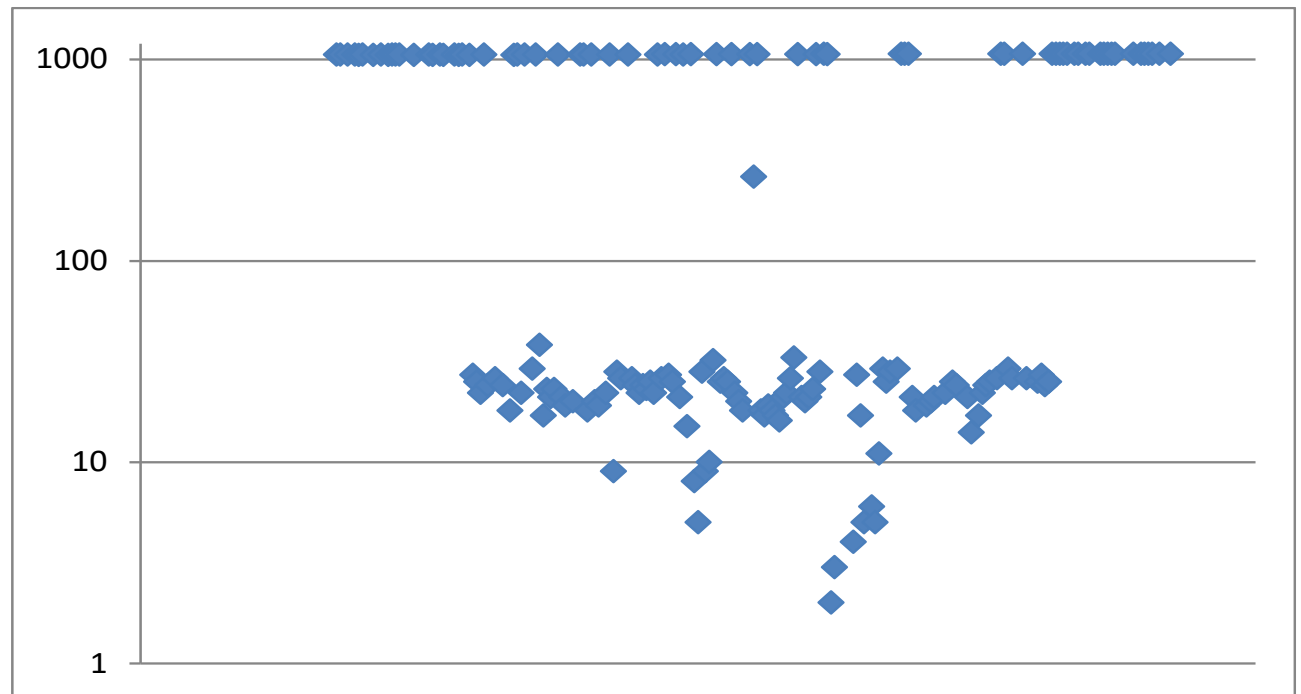
Ultrasound non-linearity

- Works linearly up to 400cm
- Goes to 1050ish when no reflection sensed
- >600 = nothing sensed

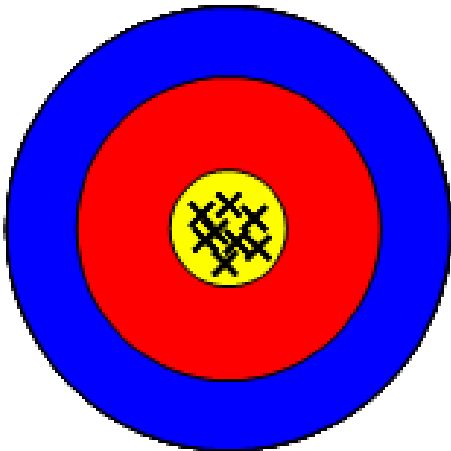
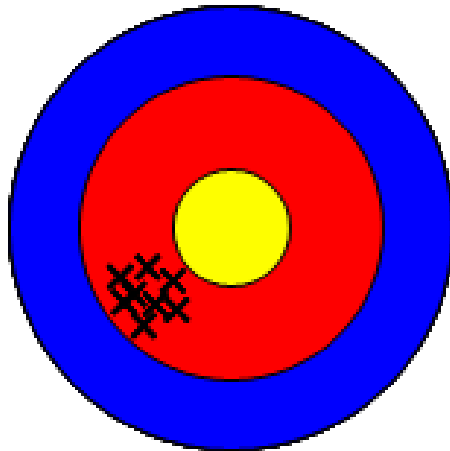
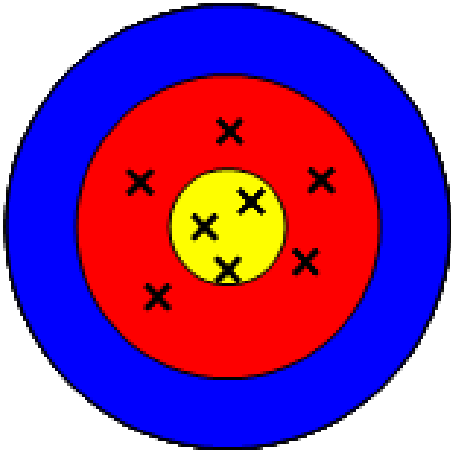
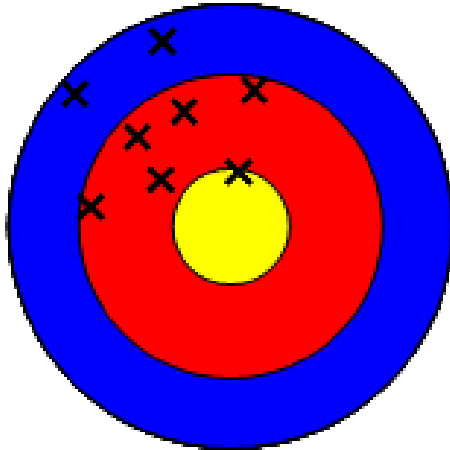


Ultrasound non-linearity

- Works up to 400cm
- Goes to 1050ish when no reflection sensed
- >600 = nothing sensed



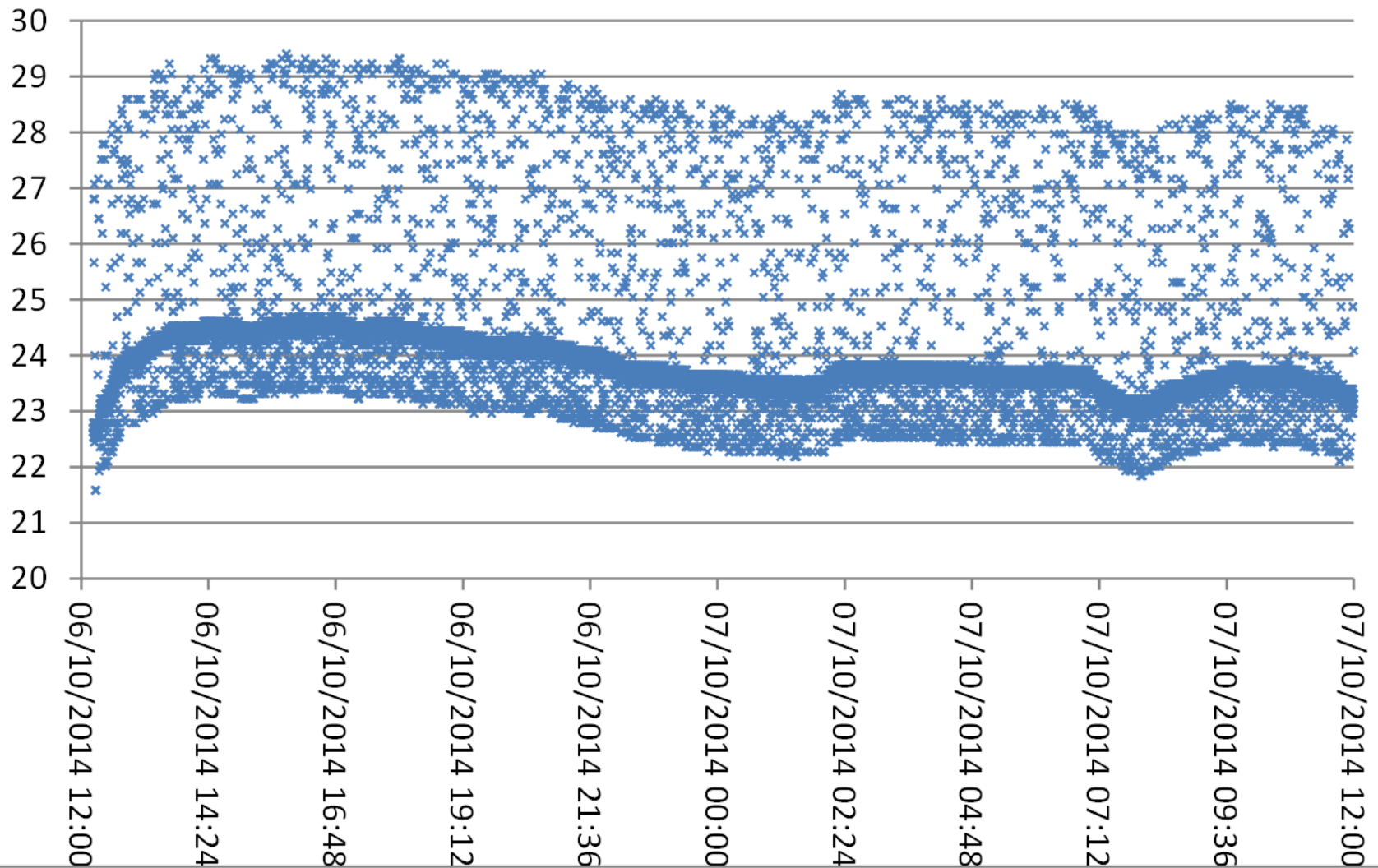
Accuracy and precision

	Accurate	Inaccurate (systematic error)
Precise		
Imprecise (reproducibility error)		

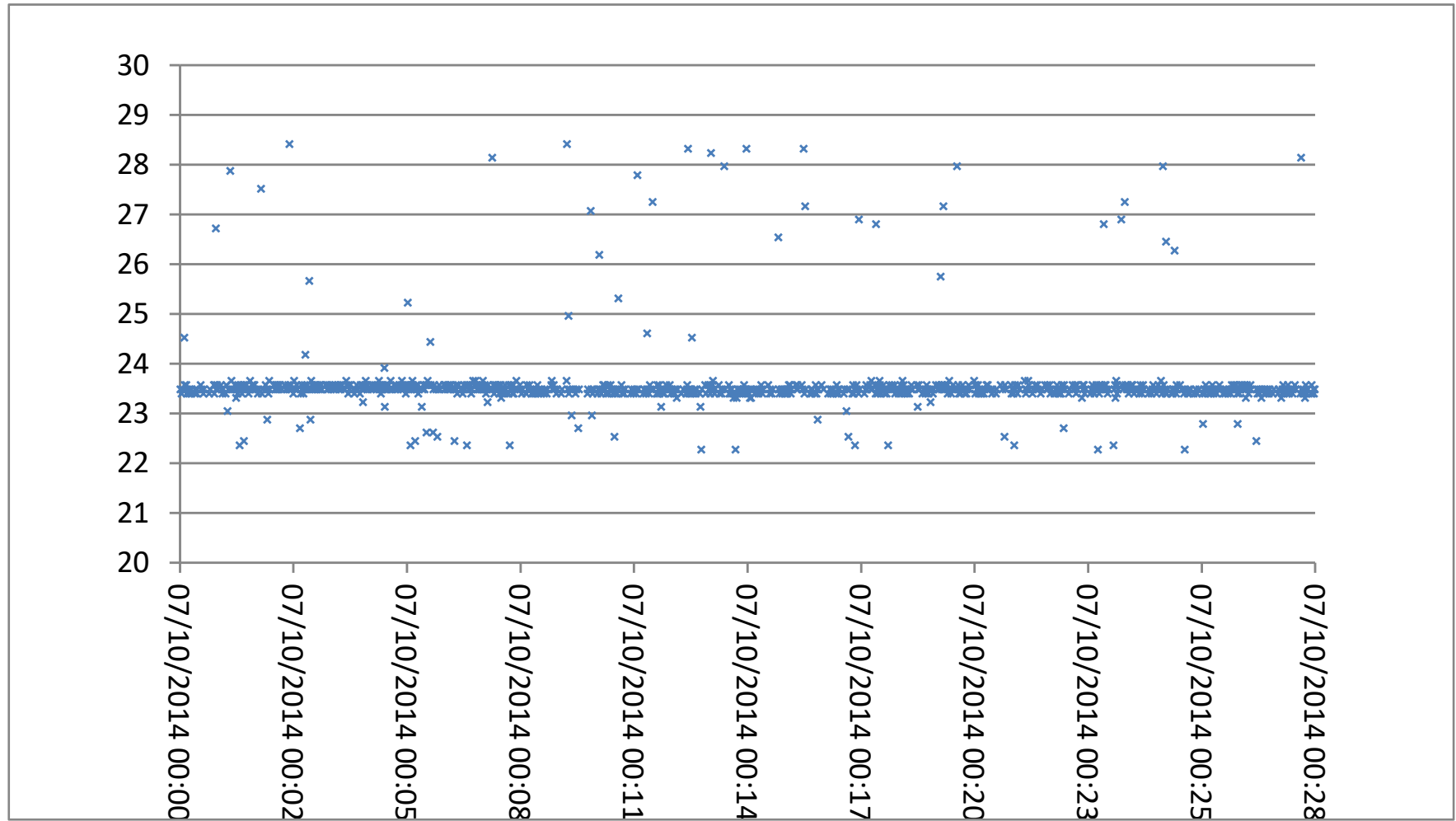
Random Noise

- Often assumed to be gaussian / normal distributed
 - Characterized by mean and standard deviation of error
- Estimating noise characteristics
 - Record n samples where you're expecting a constant value
 - Calculate mean/s.d. (from sample variance)
 - Can also estimate from slow moving data with high pass filter, but above is usually good enough
 - In practice, first step = record some data and look at it to get a rough idea of noisiness

Temperature sensor noise



Temperature sensor noise 2

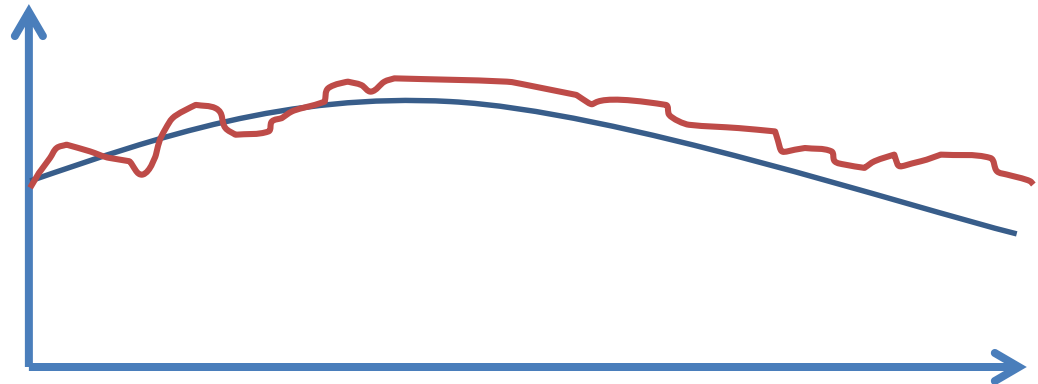


Bias and scale errors

- Bias
 - Reported value = real value + bias
- Scale Factor
 - Reported value = real value * scale factor
- Drift
 - Changes over time
 - E.g. different responses as temperature changes

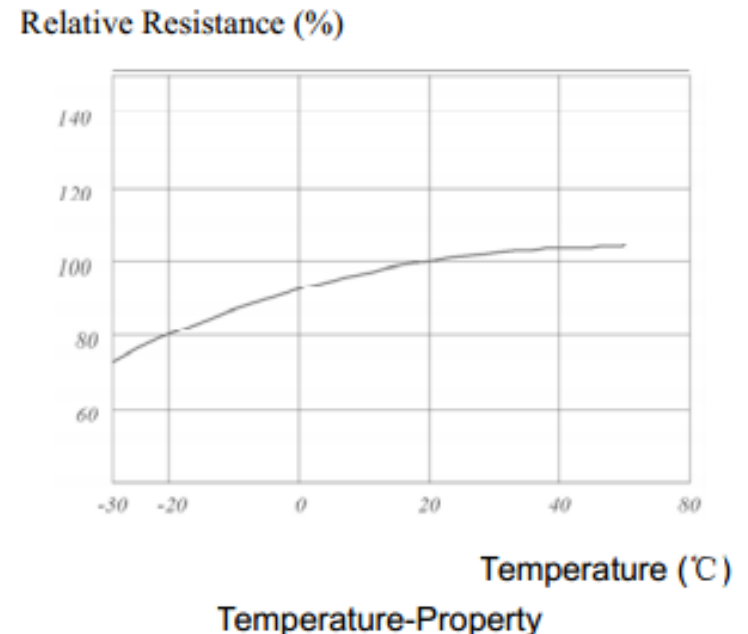
Drift

- Sensor bias or scale may change over time
- Most common when integrating sensors
 - E.g. integrate accelerometer to get speed / position = almost impossible task
- But some sensors may drift when temperature changes



Interfering Inputs

- Other things can change how a sensor responds – e.g. temperature of light sensor changes response.
- Not likely to make a massive difference on coursework, as temp changes are small.
- If temperature sensor is mounted near to hot circuitry, it can heat up.



Operator error

- Pointing a directional sensor the wrong way
- Shadowing the light sensor while testing
- Bug in code
- Forgot to plug it in
- Putting the temperature sensor right next to an air conditioning vent
 - Last year I put a sensor box in A32, by an aircon duct
 - Will respond differently to the temperature (in part a second order response)

Non numerical errors

- Presence / not presence
 - Ultrasound, PIR motion sensor
 - Not noticing something when it is there (false negative)
 - Going off when nothing is there (false positive)
 - About the physical design of the system and the nature of the sensor, e.g. are users likely to be misaligned with the beam of the sensor.
 - Can't really fix in code, but have to take into account in algorithm design
- Multi-path errors with ultrasound sensor
 - Ultrasound bounces back to the sensor via another surface
 - If an angled surface is put in front of an ultrasound sensor it can cause beams to reflect off
 - Causes odd outliers every so often

Non numerical errors 2

- Bounce
 - Button/touch sensor
 - When pressed or released doesn't always give clean 0,1 transition
 - 0,0,0,0,0,1,0,1,0,0,1,1,1,1,1,1,1,1,0,1,0,0,1,0,0,0,0
 - Not a problem if you're only fussed about whether it has been pressed, or time of first press.
 - If you want number of times pressed or to do an action on button press, need to 'de-bounce'
 - E.g. by only allowing 1 press per 0.2 seconds
 - De-bouncing can be useful if you have algorithms on other sensors which give out a 0/1 type classification

Dealing with errors

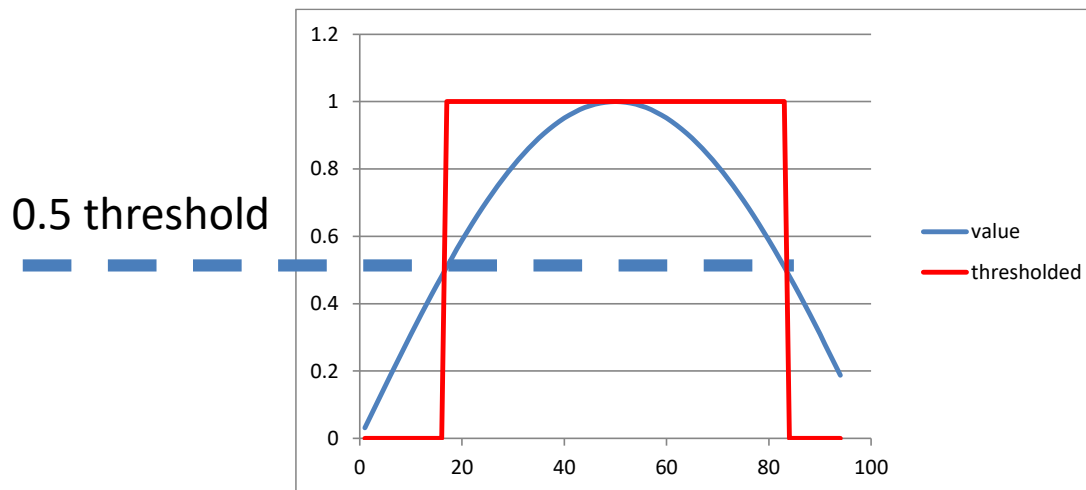
- 1) Have a look at your sensor data (load it up in excel & graph it or whatever), try and work out what type of errors there are.
- 2) Filter the data to remove as much error as you can
- 3) Design your algorithms to be tolerant of errors (e.g. detect something n times, rather than just signalling a detection the first time)

Dealing with numerical errors

- (Intelligently) Thresholding and peak finding
- Filtering / smoothing & averaging

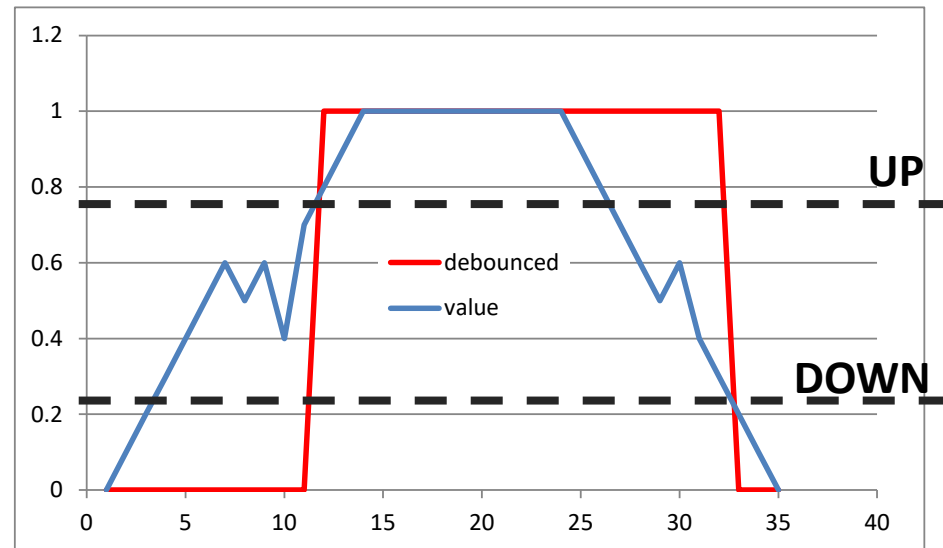
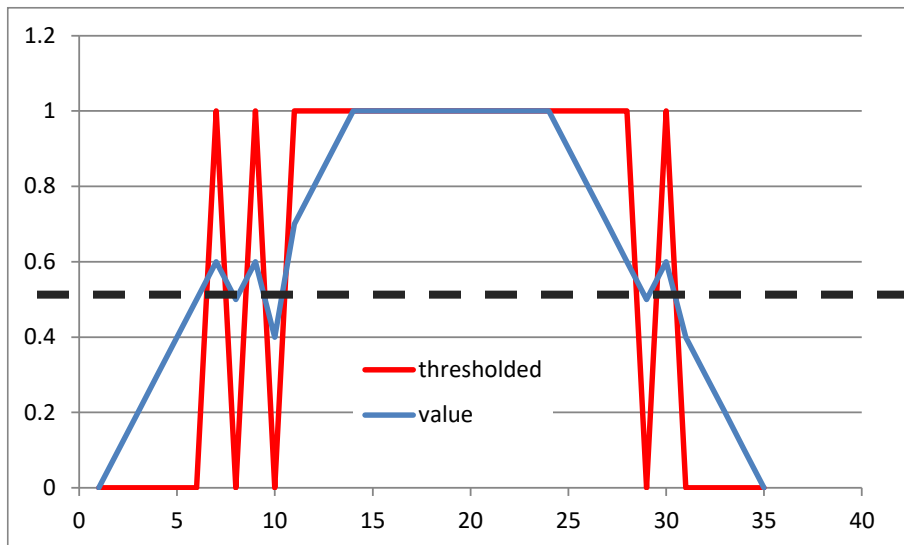
Thresholding

- Useful for:
 - A) Detecting events
 - B) Ignoring non-event noise
- Basically, pick a level, and check if the value is greater than this level.



De-bounced thresholds

- Different thresholds for up and down changes
 - Avoiding bounce, when number wiggles around close to the value



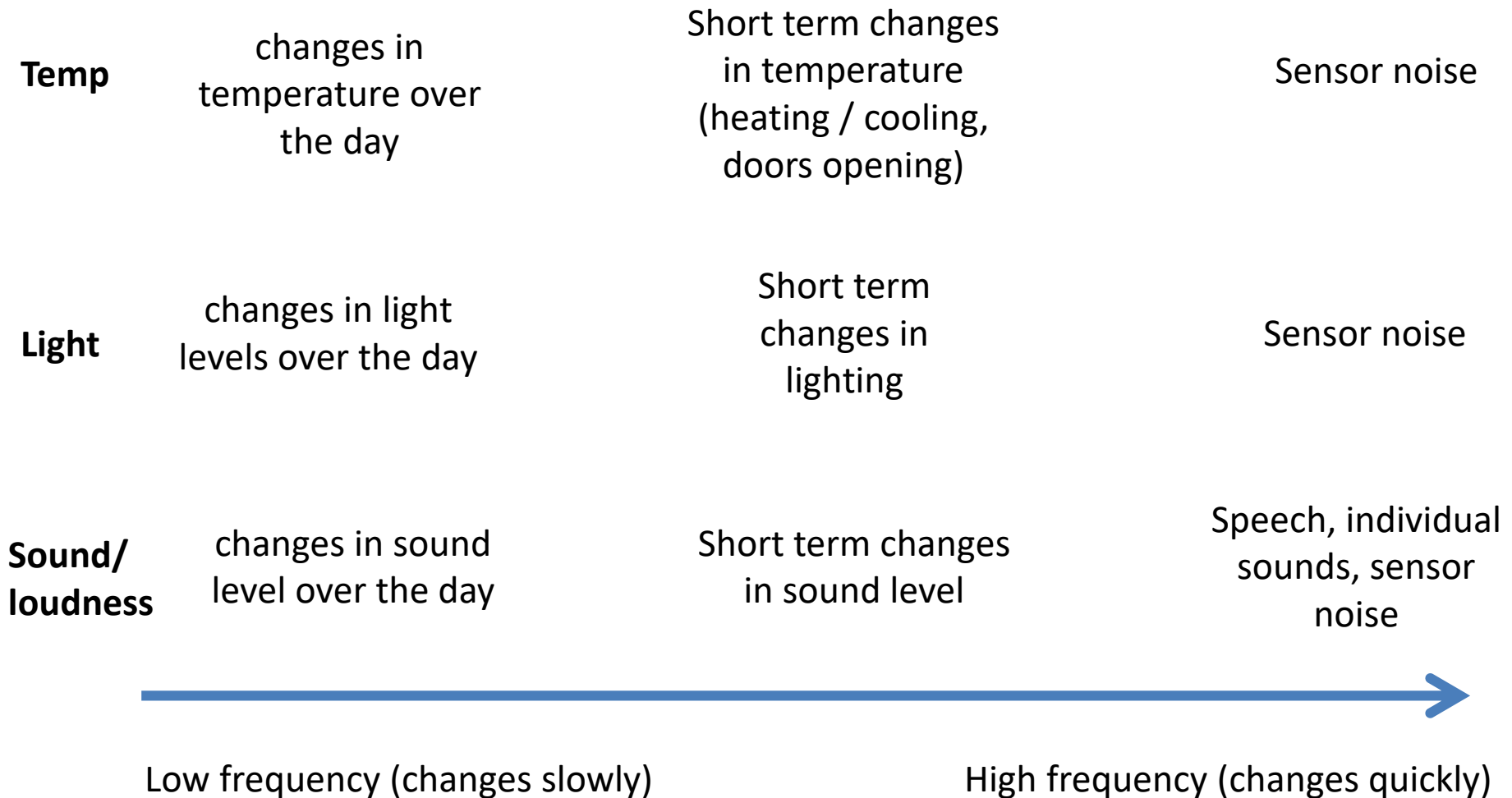
Thresholding for events

- Threshold data (maybe debounced)
- Can fire events when threshold rises or falls:
 - Can also consider how long it has been in the previous state (e.g. how long has a button been pressed)
 - Counts and also be useful, e.g. how many times has the threshold value been hit in the last minute

Adaptive thresholds

- Setting the value of the threshold based on recent data
 - e.g. ‘threshold is hit when we see a value which is 40 more than the mean of the last 10 minutes’
 - ‘threshold is hit when the value is more than 2 x the standard deviation of the last 10 minutes above the mean of the last 10 minutes’
 - i.e. threshold setting using descriptive statistics of some kind
- Often a way to avoid triggering based on long term variations or changes in sensor response due to temperature
- Can often achieve some of this by high pass filtering before thresholding (will explain high pass in a moment)

Different frequencies



Smoothing and filtering

- Allows us to identify the frequencies we are interested in
- Smoothing
 - Includes low pass filtering
 - Removes random noise
 - Leaves longer term variations
 - Low pass filters can be used to do this
 - Smoothing often does similar things, but is not frequency base
 - You are extremely likely to want to use this in your coursework
- Detrending
 - Remove long term bias and drift
 - Leave short term variations
 - But loses absolute value
 - Your algorithm must use differences
 - High pass filters can be used to do this
 - You may want to use this to implement a thresholding algorithm or something
- Some specific filtering (e.g. sound/loudness filter)
- If you are hard-core and want to implement more complex linear filters, this page can be very helpful to do the calculations for them:
 - <http://www-users.cs.york.ac.uk/~fisher/mkfilter/>
 - Beyond the scope of this course

Exponential low pass filter

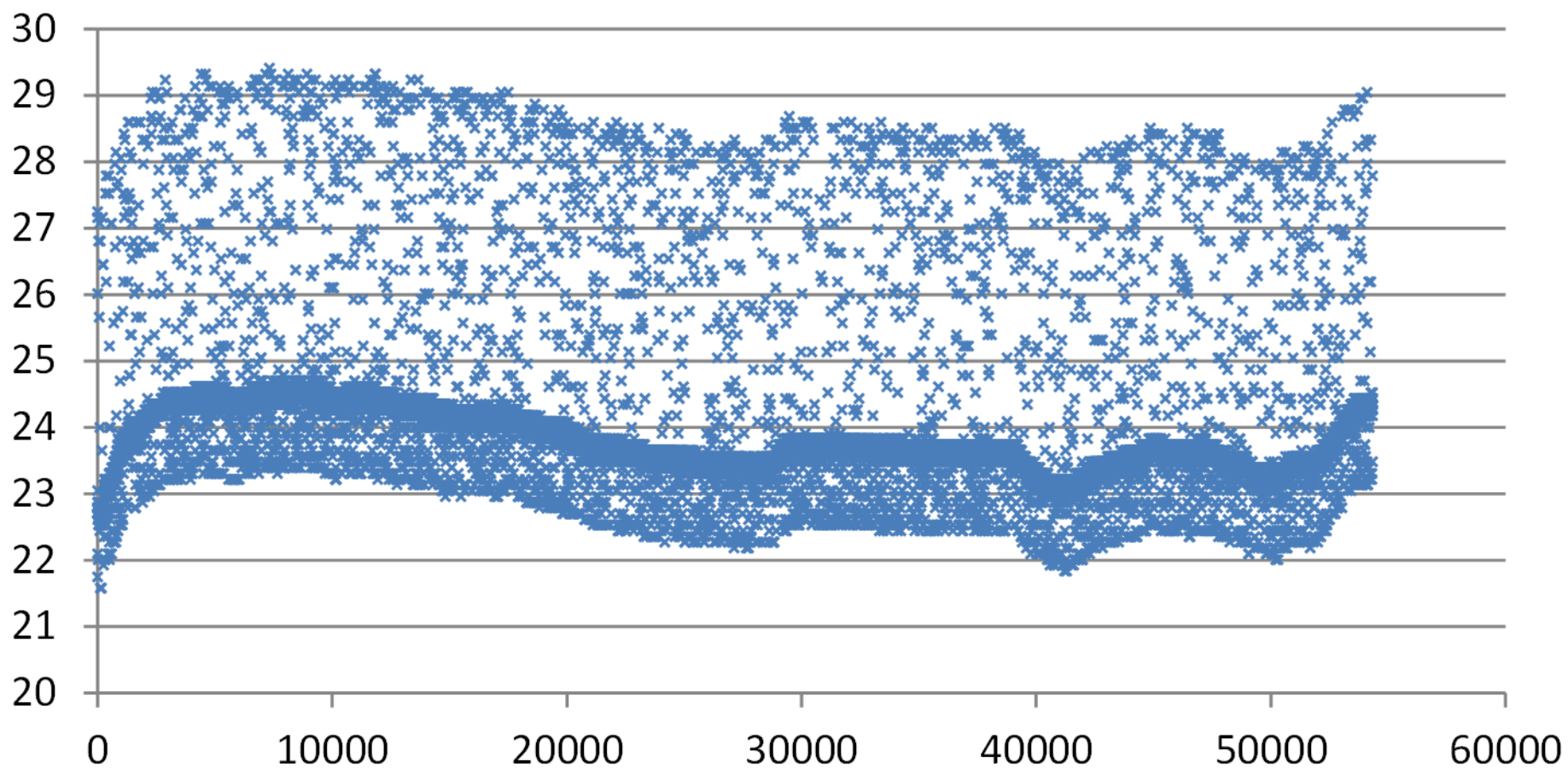
- Simple but effective smoothing filter
- Very widely used
- $y(k) = (1-a) * y(k-1) + a * x(k)$
- Filter constant a , defined by
- $a = 1 - \text{EXP}(-T/\tau)$ $a = 1 - e^{-T/\tau}$
- T = time step per sample
- τ = time constant = the time it takes to get to 0.63% if going from 0..1 (same units as T) (this is called a step response)
- Constant up -> Delay up

Exponential filter in python

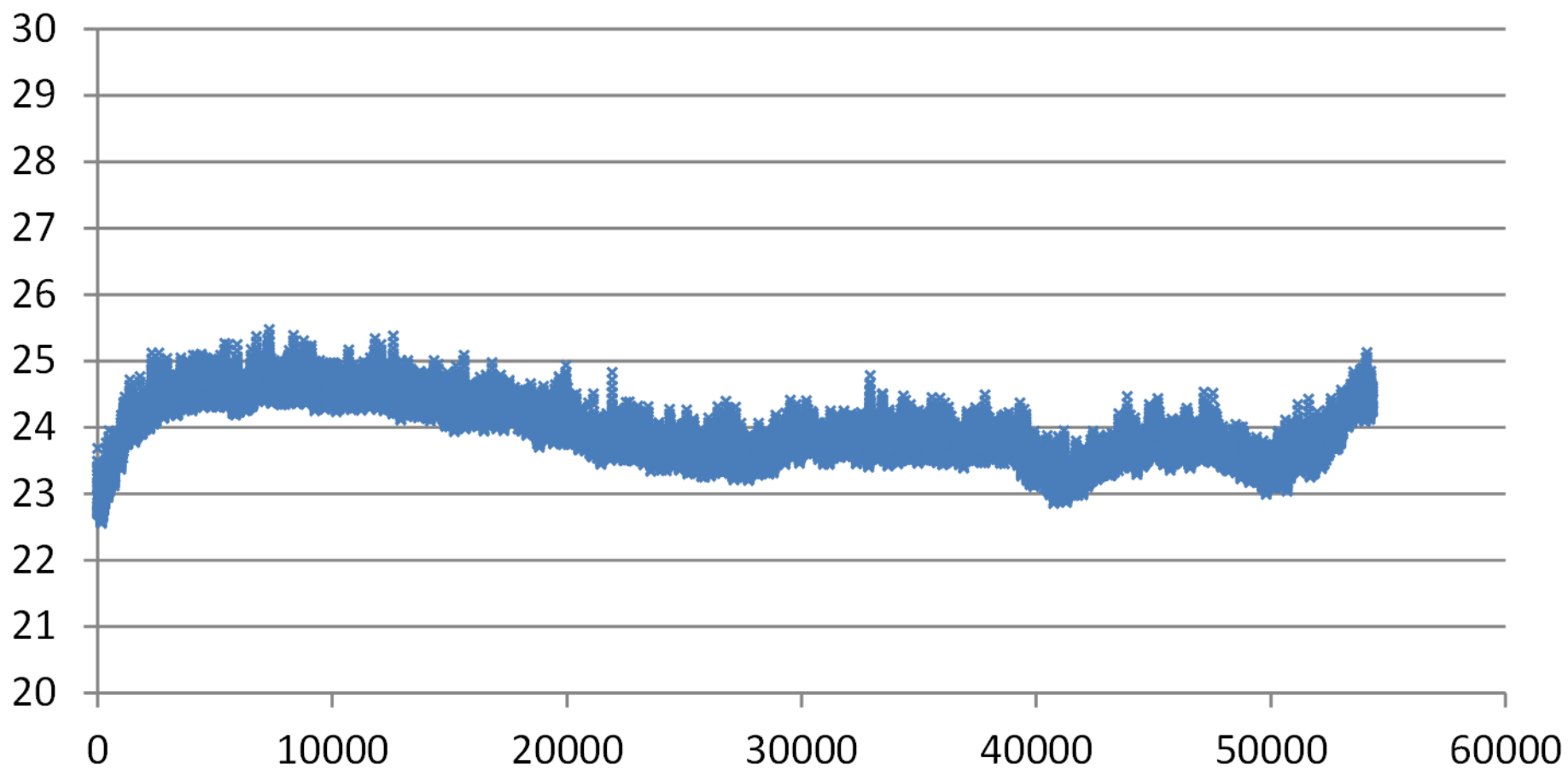
```
import grovepi
import time

lowPassed=0
constant=0.1
print("Time,Raw data, Low pass")
while True:
    value=grovepi.analogRead(0)
    lowPassed=lowPassed*(1.0-constant) + value * constant
    print("%4.4f,%4.4f,%4.4f"%(time.time(),value,lowPassed))
    time.sleep(0.05);
```

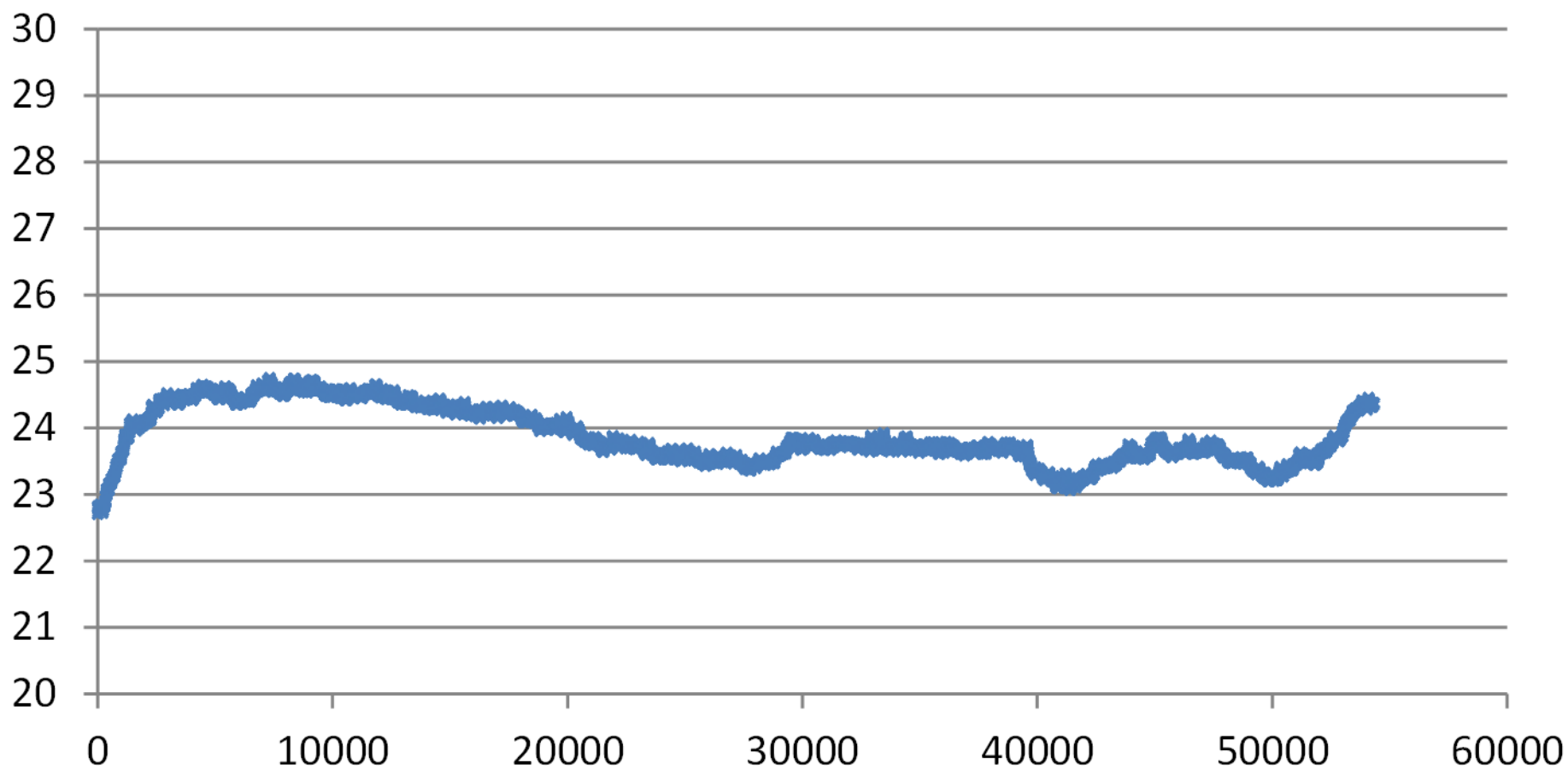
Filter constant 1



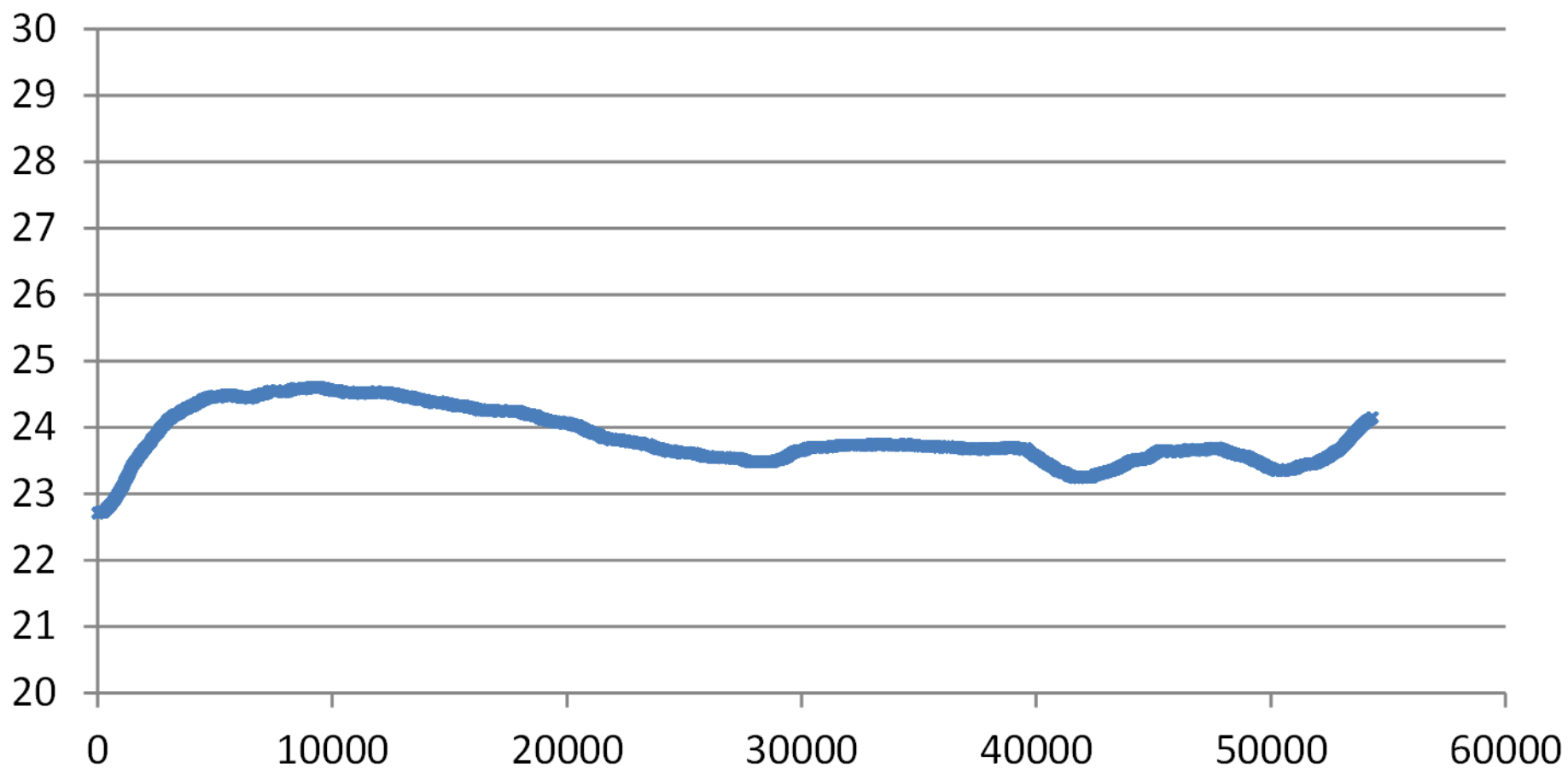
Filter constant 0.1



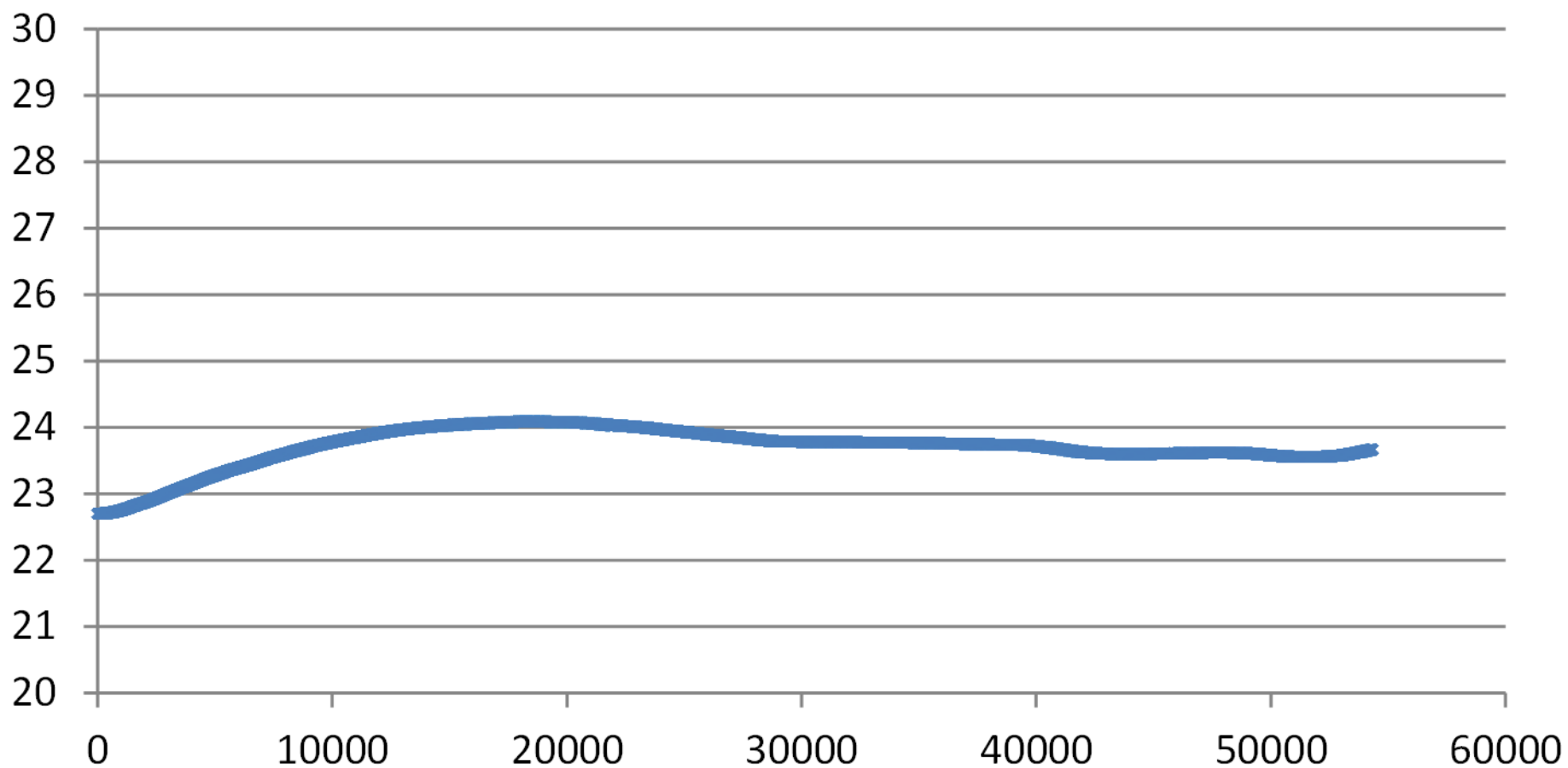
Filter constant 0.01



Filter constant 0.001



Filter constant 0.0001



De-trending / high pass filtering

- Allows you to see the short term variations in signal
- But ignoring long term fluctuations
- Loses the absolute value
- Can be good for thresholding
 - For example, want to detect temperature drops on external door opening
 - But don't want to get set off by changes over the day
 - So high pass filter to remove slow changes over the day, then set threshold on the output
 - Often useful to do both high pass and low pass filters (remove very slow changes plus remove random noise)

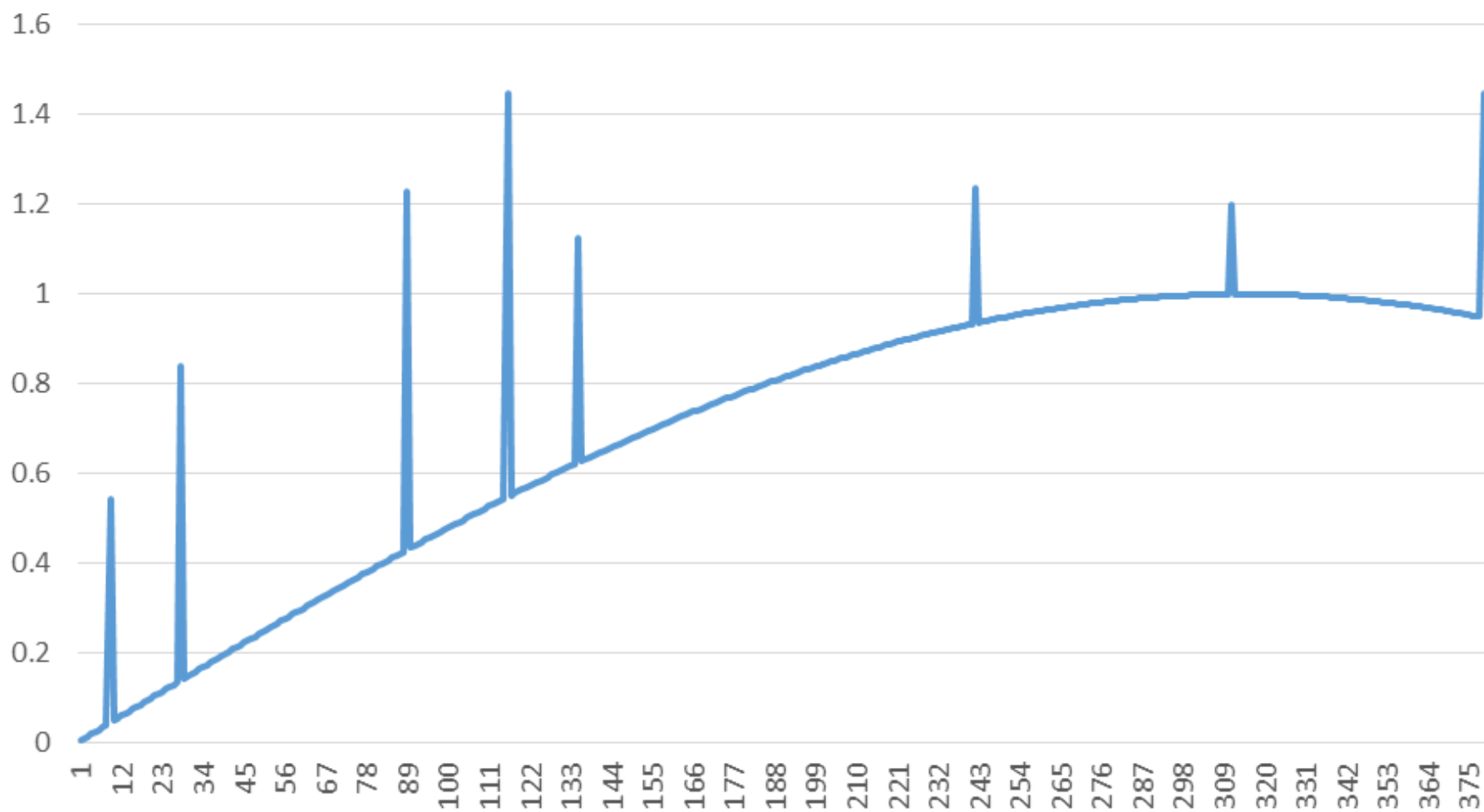
Simple high pass filters

- First order high pass
- $y(k) = a * (y(k-1) + x(k) - x(k-1))$
- Filter constant a related to time constant by:
- $a = \tau / (\tau + T)$
- T = time step per sample
- τ = time constant
- (time for filter to return to 0.37 after a step up to 1) – higher = less intensive processing.

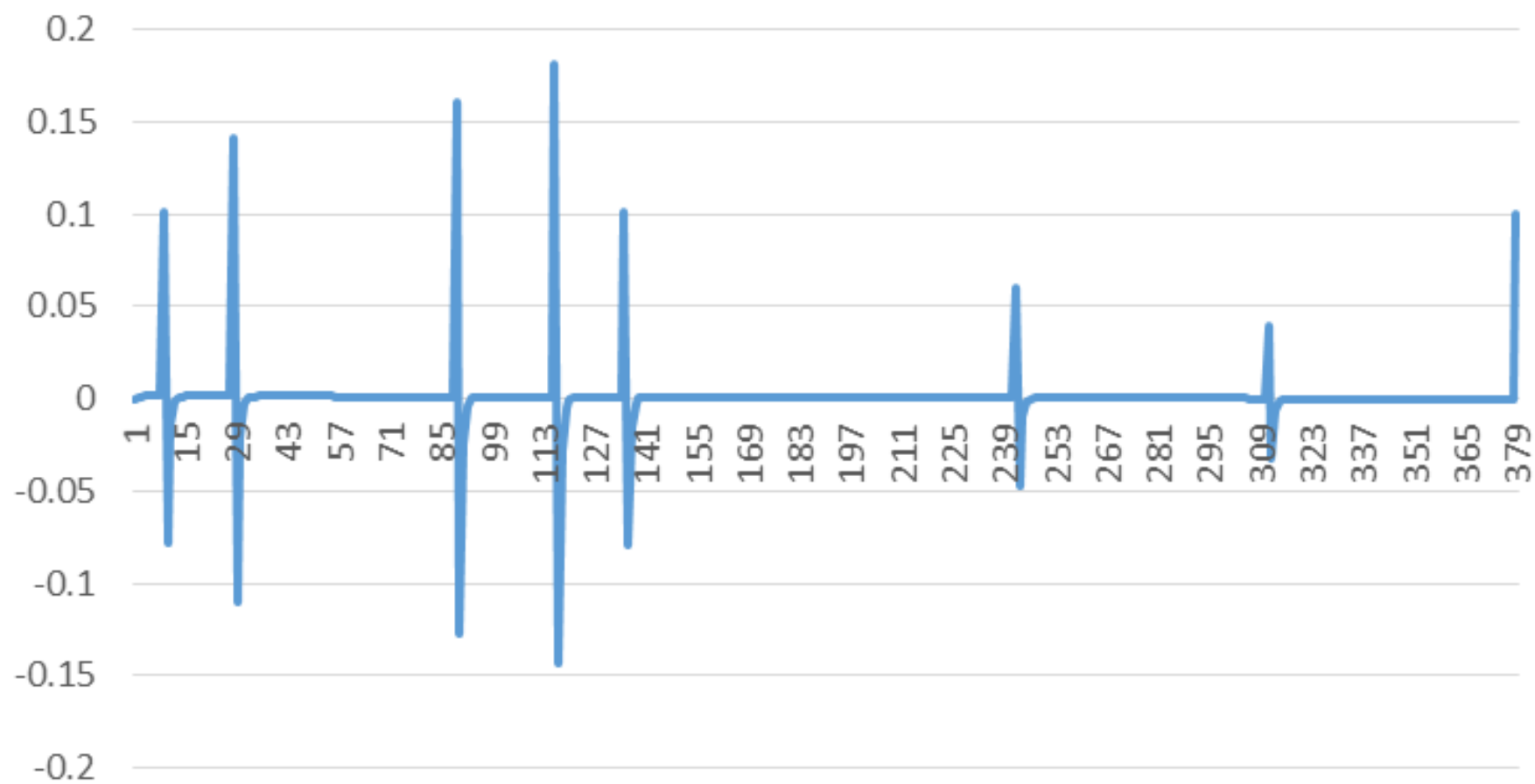
High pass python

```
import grovepi
import time
highPassed=0
constant=0.1
lastValue=0
print("Time,Raw data,High pass")
while True:
    value=grovepi.analogRead(0)
    #  $y(k) = a * (y(k-1) + x(k) - x(k-1))$ 
    highPassed=constant * (highPassed + value -lastValue)
    lastValue=value
    print("%4.4f,%4.4f,%4.4f"%(time.time(),value,highPassed))
    time.sleep(0.05)
```

Spikes with underlying slow movement



High pass filtered spikes only



Combining high and low pass filters

- If frequency of interest is somewhere in the middle
 - Remove random noise
 - But ignore any short term fluctuations
- Do low pass then high pass the output
- Called a 'band pass filter'
- There are cunning ways to implement band-pass in one coding step
 - Beyond the scope of this course again
 - <http://www-users.cs.york.ac.uk/~fisher/mkfilter/>

Median Filter

- Output a median of the last N sensor values
- Non linear filter
- Very good for removing outliers, without smoothing things out
 - Exponential filter jumps with outliers
- Good for sensors with extreme non-linearity, e.g. ultrasound, or digital sensors e.g. touch, button
 - Don't smooth ultrasound with a standard low pass filter without taking into account the special out of range values
 - With digital sensors, collapses into being really a 'mode' filter.
- Retains changes in value / step changes
- Often doesn't need many values to work well
 - So reduces delay
 - delay = half the median length

Median filtering example

```
import grovepi
import time
import collections # for deque class

# the following code does the median filter itself

# create a deque for history
# a deque is a double ended list,
# put things in the end when it is full
# (when it contains maxlen values), and things
# will be pushed off the other end
historyBuffer=collections.deque(maxlen=21)

print("Time,Raw data,Median")

while True:
    dataPoint=grovepi.analogRead(0)
    historyBuffer.append(dataPoint)
    orderedHistory=sorted(historyBuffer)
    median=orderedHistory[int(len(orderedHistory)/2)]
    print("%4.4f,%4.4f,%4.4f"%(time.time(),dataPoint,median))
    time.sleep(0.1)
```

Median filtering example (2)

```
# example code used in the lecture to make the graph of how a median filter removes  
# noise
```

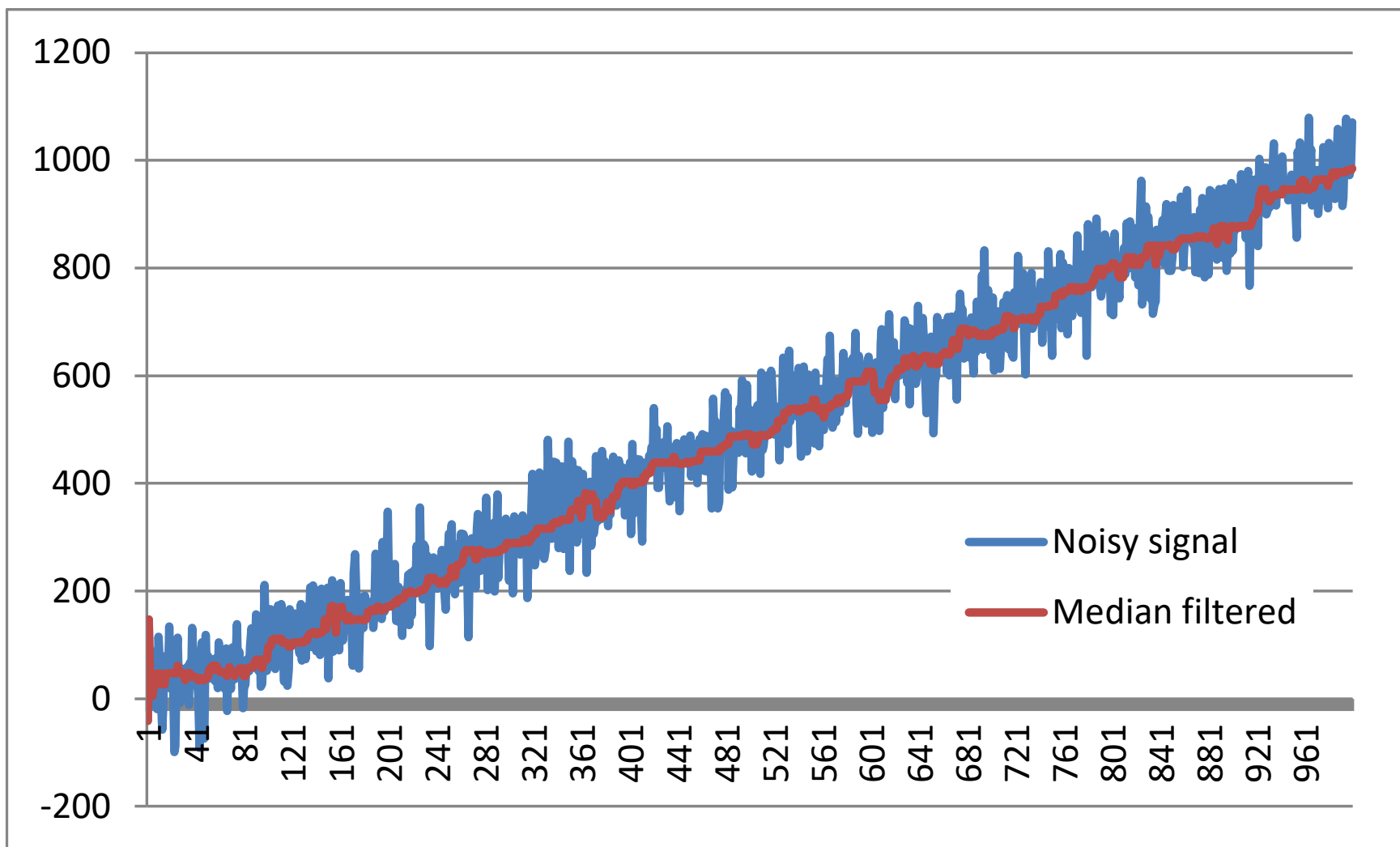
```
import random  
import collections
```

```
# generate a rising line with some random noise added to test this with  
testData=[]
```

```
for c in range(0,1000):  
    # normal / gauss random noise, variance 50 (so very noisy)  
    noiseValue=random.gauss(0,50)  
    dataValue=c  
    testData.append(dataValue+noiseValue)
```

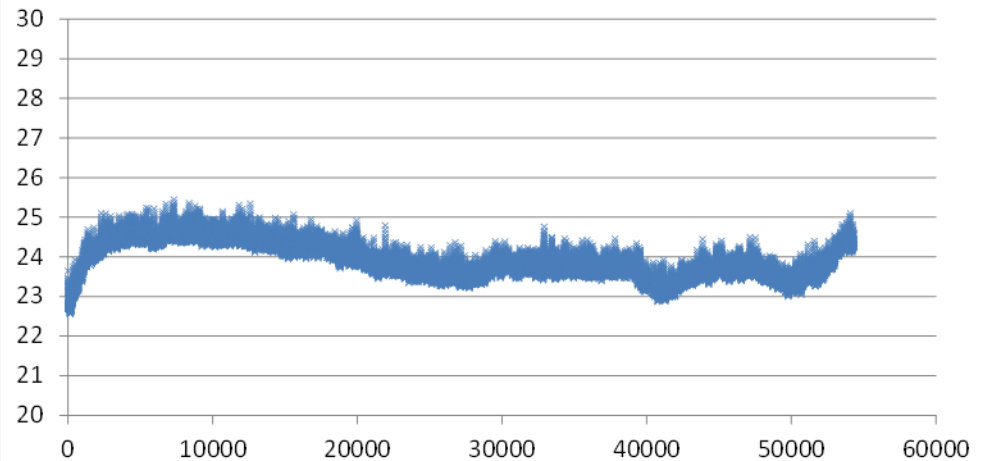
```
# the following code does the median filter itself
```

```
# create a deque for history  
# a deque is a double ended list,  
# put things in the end when it is full  
# (when it contains maxlen values), and things  
# will be pushed off the other end  
historyBuffer=collections.deque(maxlen=21)  
for dataPoint in testData:  
    historyBuffer.append(dataPoint)  
    orderedHistory=sorted(historyBuffer)  
    median=orderedHistory[int(len(orderedHistory)/2)]  
    print(dataPoint,",",median)
```

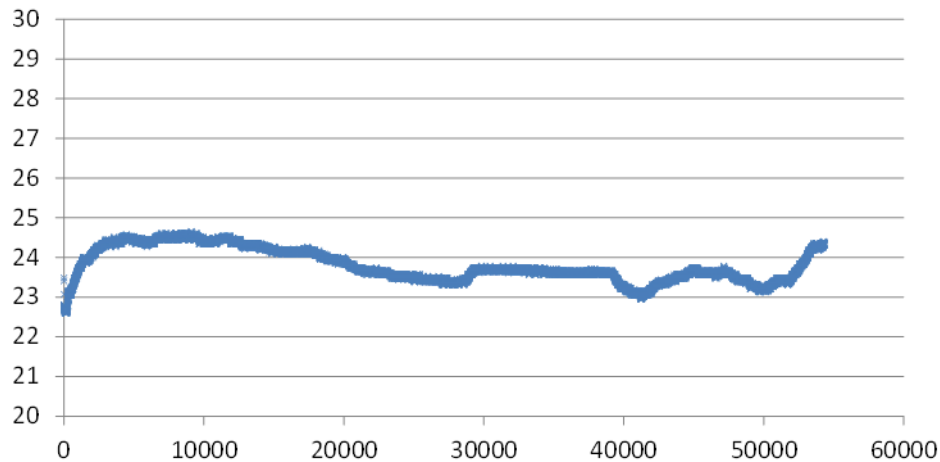


Exponential vs Median Filtering of Temperature

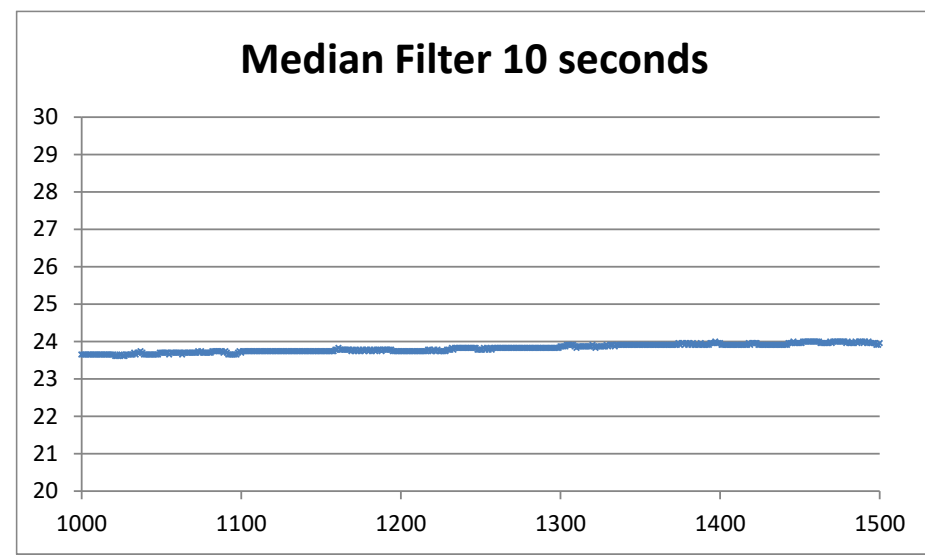
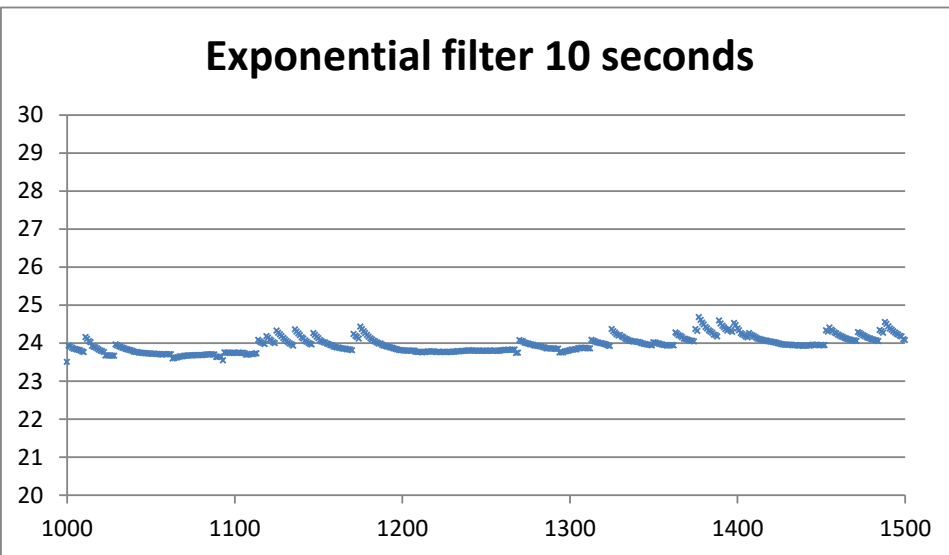
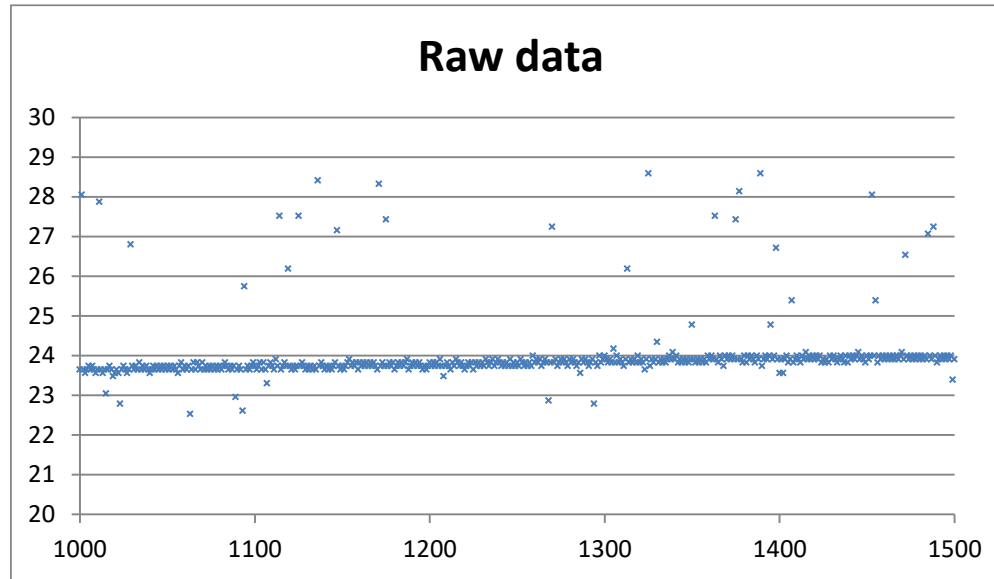
Low pass - 10 second time constant



Median Filter 10 seconds



Exponential vs median close-up

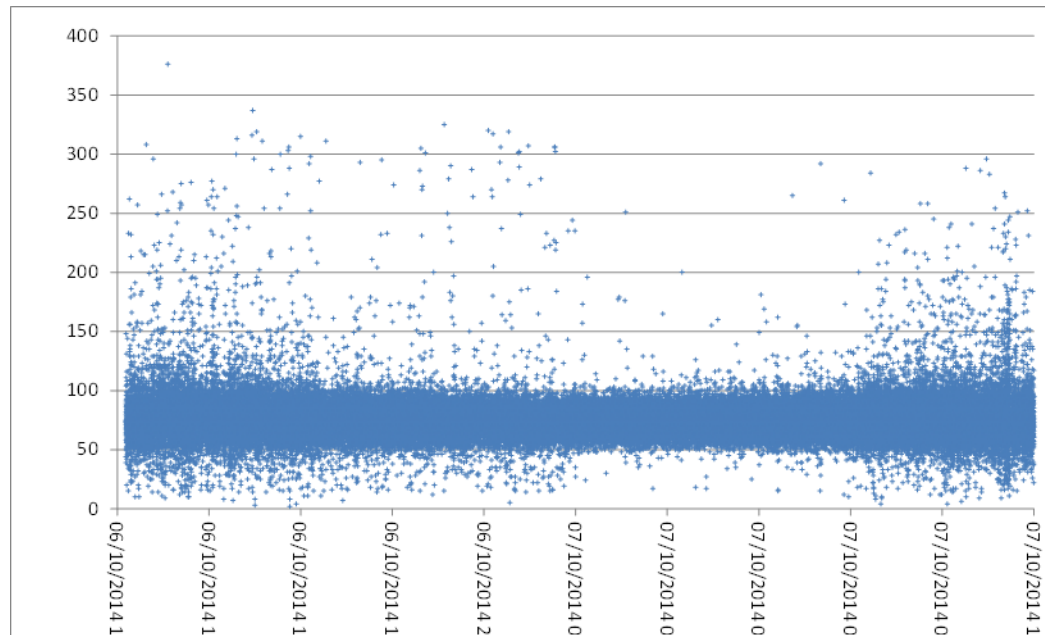


Sound / loudness sensor filtering

- Choice of loudness and sound sensor
 - Loudness sensor has more built in filtering than the sound sensor
 - Loudness = best for environmental sensing (sound level in a room)
- Audio contains multiple levels of information. You are likely to have to filter the signal to get anything even vaguely useful.
 - Individual sounds (e.g. loud bangs or taps or whatever)
 - Overall sound level

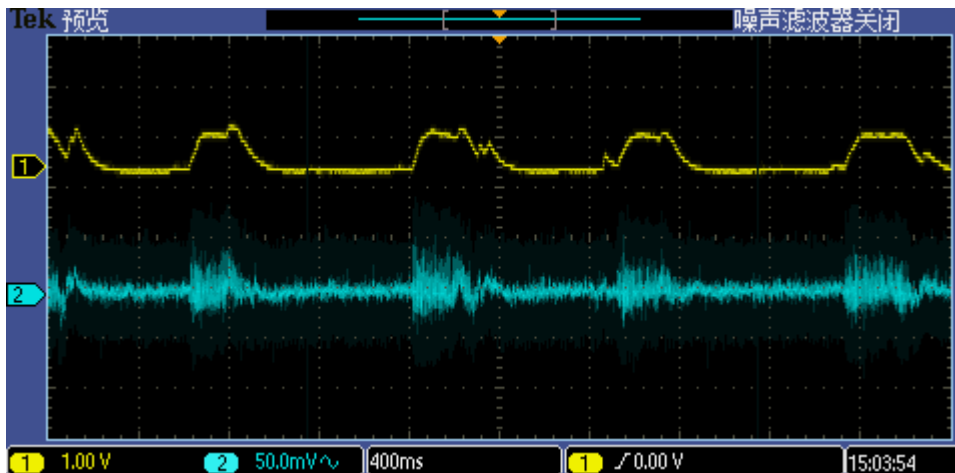
Unfiltered sound is rubbish

- If you want to get a numerical value out of it, you need to do processing
- Especially if you're using low sample rates – e.g. graph below shows 24 hours of samples from A32



Sensing overall sound level

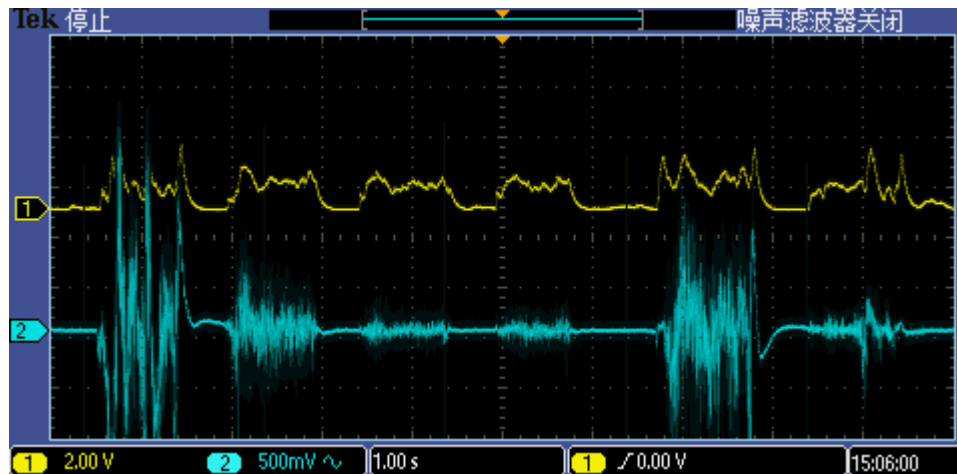
- Sensor shows something that is a (somewhat filtered) version of the raw sound envelope



- Getting an overall sound level requires filtering

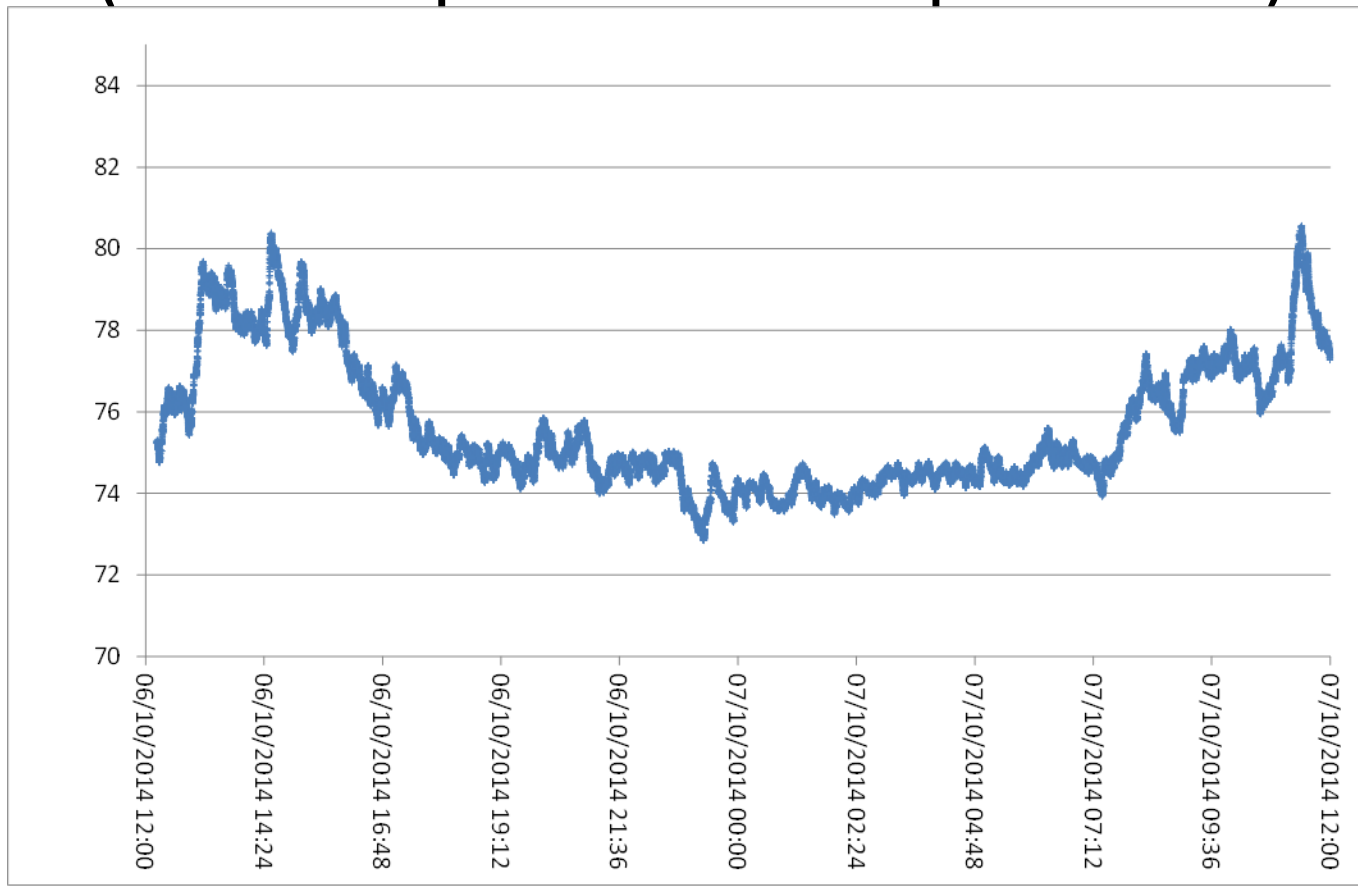
Sensing overall sound level (2)

- What do you mean by overall sound level?
 - How noisy it is **now** (or in the last x minutes)
- Can't just take one sample, because of frequencies in sound



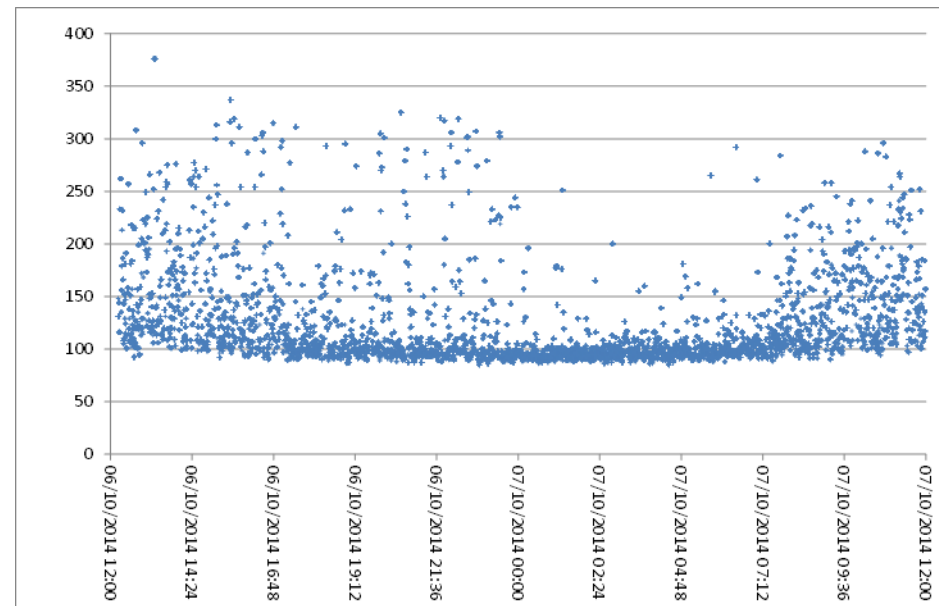
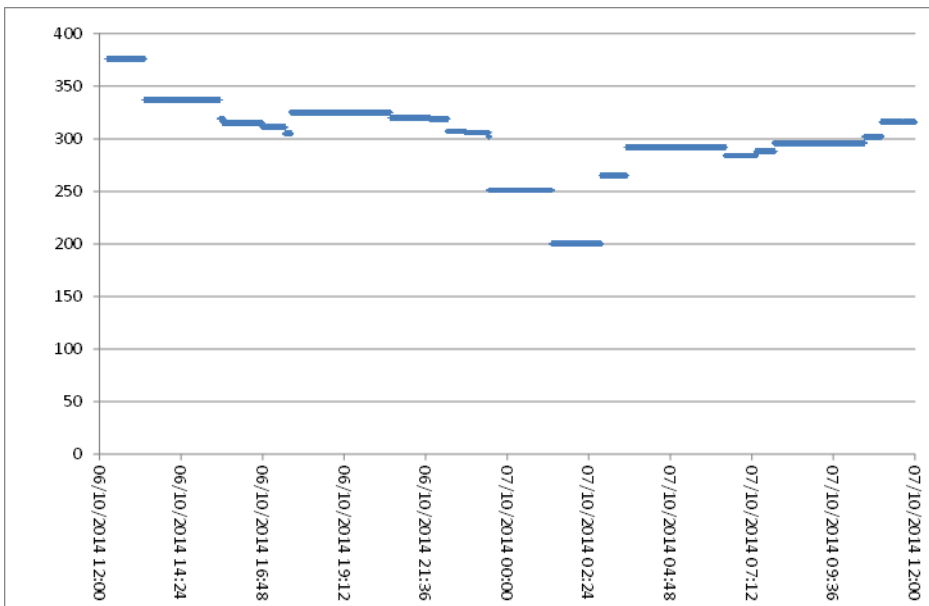
Sensing sound level averages

- How noisy can mean multiple things: One approach is to take a mean over a period of time (or use exponential low pass filter)



Sensing sound level maxima

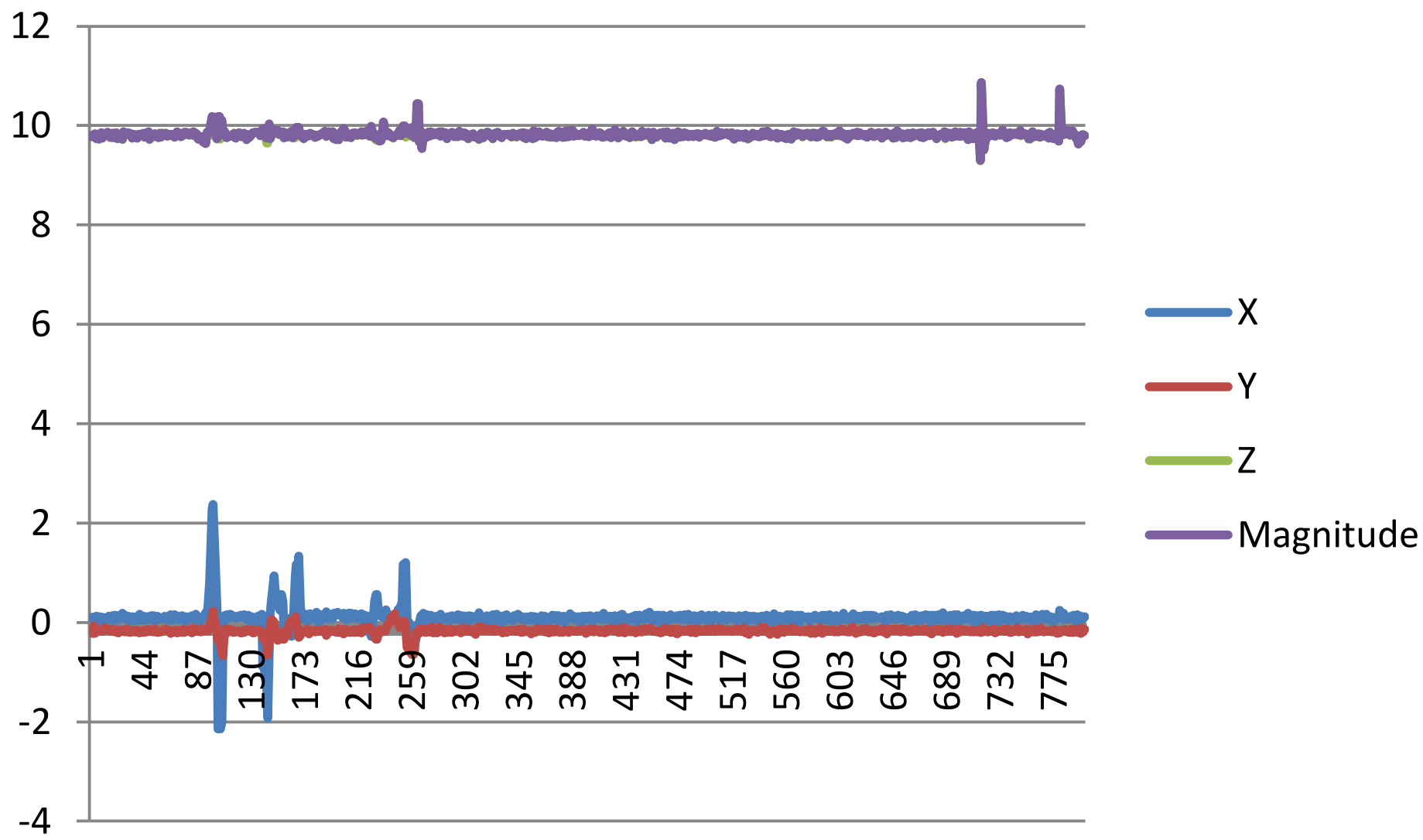
- A second approach is to look at the maximum over a period of time
- Finding the 'loudest noise in X time'
- E.g. loudest noise in an hour, minute



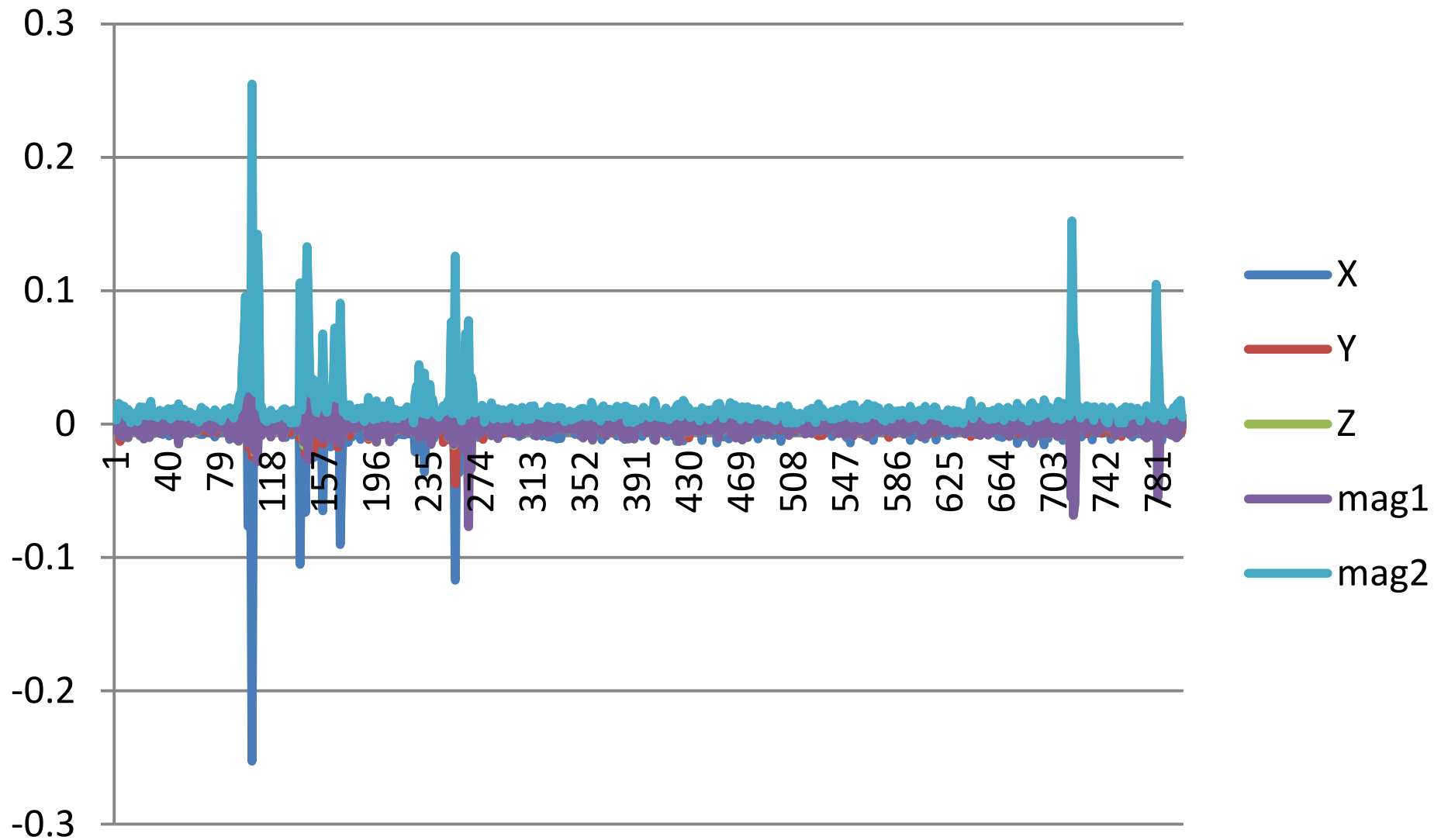
Accelerometer

- Detects acceleration on 3 axes, X, Y, Z
- That acceleration includes gravitational acceleration
 - Constant magnitude 9.81
 - Points down (so you can use the accelerometer to detect tilt)
- Event detection examples:
 - Knock: High pass filter, then threshold
 - Drop: Detect magnitude $\ll 9.81$ (0 = freefall)

Knock – raw data



Knock – high pass filtered



Take home (or into lab) messages

- Sensor data is dirty
- Sensor data contains a lot of information, and you may only want some of it
 - E.g. short or long term variations
- You are likely to need to clean up your sensor data using various methods
 - Low pass, high pass, median filtering, thresholding, debouncing
 - The vibration based nature of sound means that it requires particular filtering
- If you need help with this stuff, ask us in the labs